Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2023-IC-006

2023-IC-006

Vision-based Widget Mapping for Test Migration across Mobile Platforms: Are We There Yet?

Ruihua Ji, Tingwei Zhu, Xiaoqing Zhu, Chunyang Chen, Minxue Pan, Tian Zhang

Technical Report 2023

Vision-based Widget Mapping for Test Migration across Mobile Platforms: Are We There Yet?

Ruihua Ji[†], Tingwei Zhu[†], Xiaoqing Zhu[†], Chunyang Chen[‡], Minxue Pan[†*], Tian Zhang[†*]
[†]State Key Laboratory for Novel Software Technology, Nanjing University, China
[‡]Monash University, Australia
yiting.ji@gmail.com, {tingweizhu33, xqzhu}@smail.nju.edu.cn,
chunyang.chen@monash.edu, {mxp, ztluck}@nju.edu.cn

***Abstract*—Automated GUI testing through the reuse of existing tests has recently gained prominence in research. Cross-platform migration of GUI tests between different platform versions of an application offers a promising opportunity for test reuse. Widget mapping, identifying similarities between source and target application widgets and connecting semantically analogous pairs, is central to these approaches. Vision-based widget mapping approaches are supposed to provide platform-agnostic solutions more suitable for cross-platform migration, considering that different platform versions frequently display strong resemblances in the appearance of their semantically similar widgets. However, the efficacy of vision-based widget mapping for cross-platform migration remains limited and the reasons remain unclear.**

In this paper, we present the first comprehensive investigation of vision-based widget mapping for cross-platform GUI test migration. We devote considerable effort to constructing a dataset consisting of 6,730 bi-directional mapped widget pairs across the iOS and Android platforms, and categorize the mapped widgets into eight classifications to thoroughly assess the capabilities of various approaches. We implement 89 configurations, derived from five distinct vision-based widget mapping methodologies, and evaluate their performance utilizing our dataset. Our findings reveal valuable insights that can be employed to advance vision-based widget mapping techniques: (1) The current approach exhibits potential for improvement, as certain configurations demonstrate superior performance in comparison to existing methods; (2) Some features can adversely impact the mapping, requiring more consideration; (3) A substantial proportion of mapped widgets display varying inconsistent contents in their appearance, which require more sophisticated vision algorithms.

***Index Terms*—Test migration, GUI testing, vision-based GUI analysis**

I. Introduction

GUI tests play a crucial role in ensuring the quality of mobile apps by simulating user interactions with the app's graphical user interface. However, manual test development is time-consuming, and automated test generation techniques [1]–[3] may miss certain scenarios, resulting in undetected faults related to specific app functionalities [4], [5]. To overcome this challenge, researchers propose reusing meaningful GUI tests from a source app to generate new tests for a target app, which has been shown feasible for the automated testing of Android apps within the same domain [6]–[9]. Cross-platform app development is prevalent but requires distinct GUI tests for each platform, which is redundant. Therefore, migrating GUI tests across different platform versions of an app is a potential solution for test reuse and cross-platform testing enhancement.

GUI test reuse is possible due to the semantically similar functionalities shared by many GUI applications. In the case of cross-platform mobile apps, this similarity can be even more pronounced, with identical or nearly identical functionalities and consistent appearances across platforms. This consistency facilitates users switching between platforms and enhances app retention. Based on the observation that apps provide the same or almost the same functionalities through semantically similar events, the migration of GUI tests across platforms involves mapping such events across platforms. Since an event is triggered by user interaction with a widget and this interaction can be reused as-is [10], the main challenge is to map semantically similar widgets across platforms.

The widget mapping approach for test migration can leverage consistencies in appearances and employ vision-based GUI analysis guidance instead of textual information, which includes text extracted from the source code of applications [7]–[9], [11] and platform-dependent attributes found in layout file (e.g., iOS's *label* and Android's *content descriptor*) [6], [10]. Accessing and interpreting these textual information can be challenging and platform-dependent. In contrast, vision-based approaches only capture and utilize text displayed on screens, which is extracted using optical character recognition (OCR) techniques [12]. Vision-based approaches offer a broader perspective, and provide supplementary guidance beyond textual information, thereby improving the ability to semantically map widgets. Additionally, vision-based widget mapping techniques are platform-agnostic, enhancing their applicability for test migration across platforms. Despite these advantages, research on vision-based widget mapping approaches for GUI test migration is limited. Currently, MAPIT [10] is the only approach utilizing visual features in app appearances to map semantically similar widgets for test migration across platforms, albeit with preliminary usage of visual features. TestMig [11] is another recent approach but it relies on app source code to map widgets and transfers GUI tests from iOS to Android. Other works that reuse tests across apps within the same domain also mostly rely on

[*]Corresponding authors.

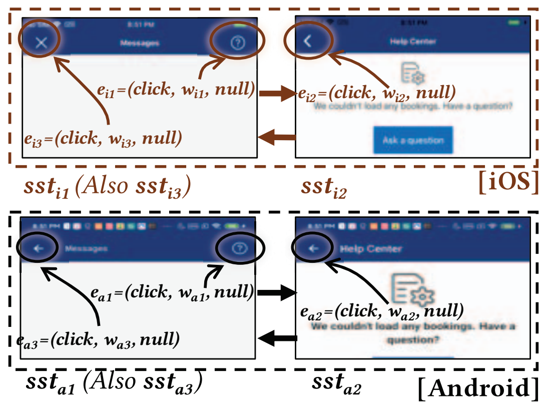
source code and textual information [7]–[9]. A recent study by Mariani et al. [6] examines the mapping of GUI events using textual information for test reuse across applications within the same domain. The study acknowledges the significance of widget mapping but does not investigate visual features. Thus, there is still a significant gap in vision-based widget mapping for GUI test migration across platforms that requires further exploration.

This paper presents the first investigation into solutions for vision-based widget mapping in GUI test migration across platforms. Given the scarcity of related work and suitable datasets, considerable effort are devoted to collecting mapped widgets from a selection of highly popular apps that are available on both the iOS and Android platforms. This enables us to construct the first real-world dataset, comprising 6,730 distinct widget mapping relations from 50 pairs of apps. We also manually identify inconsistencies in the appearances of mapped widgets based on our dataset, to comprehensively understand widget mapping for cross-platform GUI test migration and deepen our evaluation on the adopted feature comparison and matching algorithms for all kinds of consistencies and inconsistencies. It also provides guidance for further vision-based widget mapping in other related domains. Furthermore, we explore several domains of GUI testing and select five approaches, which we adapt as vision-based widget mapping approaches. We identify three key steps in the mapping process and evaluate the impact of different choices for each step on widget mapping effectiveness using our dataset. Specifically, we implement 89 configurations using the five selected approaches, and examine the 6,730 unique widget mapping relations twice, once mapping iOS widgets to Android widgets and once vice versa.

Our study reveals limitations in existing approaches and presents valuable insights for improving vision-based widget mapping methods. The key findings include: (i) A notable proportion of semantically similar widgets display inconsistent appearances, necessitating the consideration of additional features and the application of more sophisticated comparison techniques. (ii) Numerous vision-based widget mapping configurations, derived from existing techniques across various domains, surpass the performance of current approaches, revealing significant potential for enhancement. (iii) The selection of visual features demands a judicious consideration to achieve positive outcomes, as incorporating more features does not always lead to better results and certain features should be utilized in conjunction with others. (iv) Specific features, such as *text*, offer benefits only for particular widget types and should be applied cautiously. (v) Current graphic comparison algorithms associated with the graphic feature encounter difficulties handling graphical inconsistencies, necessitating further development and improvement.

In summary, this paper presents the following contributions:

- Dedicates significant effort to the creation of a real-world dataset for assessing widget mapping in cross-platform GUI test migration, making it publicly accessible for future research [13];
- Conducts the first comprehensive study of vision-based widget mapping approaches, delineating their main process;
- Evaluates 89 configurations of vision-based widget mapping, identifying configurations that outperform all the original methods;
- Examines the impact of each step in the widget mapping process on mapping outcomes, elucidating the optimal choice for each step.



e_{i1}=(click, w_{i1}, null)
e_{i2}=(click, w_{i2}, null)
e_{i3}=(click, w_{i3}, null)
sst_{i1} (Also sst_{i3})
sst_{i2}
[iOS]
e_{a1}=(click, w_{a1}, null)
e_{a2}=(click, w_{a2}, null)
e_{a3}=(click, w_{a3}, null)
sst_{a1} (Also sst_{a3})
sst_{a2}
[Android]

Fig. 1. The GUI tests for Booking on iOS and Android platforms.

II. Vision-based Widget Mapping

This section introduces the vision-based widget mapping.

A. Preliminary

This paper focuses on the investigation of Graphical User Interface (GUI) testing of mobile applications (*app*). An app consists of numerous *screens* that form its GUI. Each screen contains various visual and interactive elements, i.e., *widgets*. Our focus lies on *events* arising from user interactions (uia) with a widget (w) on the active screen, e.g., tapping a button or entering text into a field. We represent an event e as a tuple (uia, w, txt), where txt can be empty. We capture changes in the app's screens using *screenshots*, which show the state of the app. A *GUI test* is a sequence of events $\langle e_1, ..., e_n \rangle$ on screens. Its execution is a sequence of state transitions of the app from one screenshot to another (e.g., $sst_i \xrightarrow{e_i} sst_{i+1}$).

Automatic GUI tests migration across mobile platforms is to transfer tests from one platform's implementation of an app to another platform's implementation of the same app (e.g., from iOS to Android). Fig. 1 shows two GUI tests and their executions of the popular app, Booking, on Android and iOS platforms. Staring from the iOS app with $\langle e_{i1}, e_{i2}, e_{i3} \rangle$ and $sst_{i1}, sst_{i2}, sst_{i3}$, the test cross-platform migration aims to sequentially construct $\langle e_{a1}, e_{a2}, e_{a3} \rangle$ using sst_{a1}, sst_{a2}, and sst_{a3} to generate the GUI test for the Android app. The mapped widgets that trigger semantically similar events generally exhibit consistent appearance, although some inconsistencies may exist, e.g., graphics.

Identifying the mapped widgets is crucial for event matching and has a significant impact on cross-platform GUI test

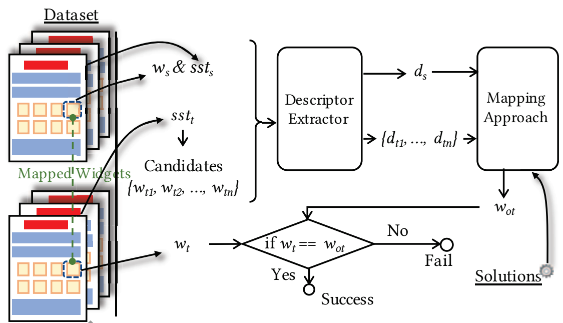


Fig. 2. The process to study vision-based widget mapping. Descriptor Extractor denotes *Vision-based Widget Descriptor Extractor*, and Mapping Approach denotes *Vision-based Widget Mapping Approach*. A widget mapping relation from DataSet provides the source and target widgets (w_s & w_t), along with the screenshots displaying the source and target widget (sst_s & sst_t). sst_t displays a set of candidate widgets ($\{w_{t1}, \ldots, w_{tn}\}$). Descriptor Extractor generates the descriptors of w_s and $\{w_{t1}, \ldots, w_{tn}\}$, which are denoted as d_s and $\{d_{t1}, \ldots, d_{tn}\}$. The output widget from Mapping Approach is denoted as w_{ot}. Solutions are configurations under test.

migration. We focus on vision-based widget mapping, which aims to exploit the consistency in widgets' appearances and mitigate the impact of inconsistency to achieve the tasks.

B. Vision-based Widget Mapping

Current approaches employ widget features as *descriptors* to represent the widgets and identify their semantically similar counterparts across different platform versions of an app. In this paper, we focus on the *vision-based widget mapping approaches* that rely on visual characteristics to generate descriptors and identify widget mapping relations. By exploiting the consistency in widget appearances and mitigating the impact of inconsistency, these approaches are able to effectively use visual features to improve widget mapping.

We illustrate the process of vision-based widget mapping in Fig. 2, building on prior research in this field. It consists of two main modules: the *Vision-based Widget Descriptor Extractor*, and the *Vision-based Widget Mapping Approach*. The former extracts descriptors of the source widget and all the candidate widgets, and the latter uses descriptors to calculate the output widget, which is verified based on the truly mapped widget.

Vision-based Widget Descriptor Extractor receives a source widget and a set of candidate widgets to identify the semantically similar widget (i.e., the target widget), along with screenshots of the source widget and all candidate widgets. It extracts descriptors by selecting one or more visual features. We review the related research work and collect the possible visual features. The features are listed as follows.

- ***text*** refers to textual content that appears as a visual feature in widget appearances. *text* can be captured using Optical Character Recognition (OCR) algorithms [10], [14].
- ***graphic*** refers to the visual content that makes up a widget's appearance, such as icons, images, and text. This means that a widget having textual content in its appearance generally can have both *graphic* and *text*.
- ***location*** refers to the position of a widget within a screenshot and is often represented as a 2D coordinate indicating the top-left corner of the widget. Such *location* is absolute location. It is commonly used, although some approaches also create *relative location* to gain deeper insights.
- ***size*** refers to the rectangular area bounded by the top-left and bottom-right coordinates of the widget in a screenshot.

The current techniques for widget mapping lack the ability to differentiate between visual and non-visual *text*, resulting in the integration of non-visual text sources such as platform-dependent XML layouts files into their *text* features, which may not be suitable for less well-known platforms. This paper prefers the visual *text*.

Vision-based Widget Mapping Approach receives descriptors of both the source widget and the candidate widgets. It compares descriptors of the source widget and all the candidate widgets, which consist of features constructing the descriptors, and then match and output the most semantically similar widget among all the candidates. This process typically involves two steps. (i) Compare each adopted visual feature individually using proper *visual feature comparison algorithms* . Various algorithms are used for comparison, such as Word2Vec [15] for feature *text* and SIFT for feature *graphic*. For a given source widget and all candidate widgets, the comparison algorithms for a specified feature may output a similarity value list or directly identify the most semantically similar widget in terms of that feature. (ii) Design a *matching method* to determine the most semantically similar widget based on the comparison results from Step (i). If the descriptor adopts more than one feature, the *matching methods* aggregate all the comparison results properly to arrive at a final decision of the most semantically similar widget.

III. Empirical Study Design

This section provides the design of our investigation into vision-based widget mapping, including an overview, our real-world dataset, the selected techniques, the research questions, and the experiment design and settings.

A. Overview

This study employs the process illustrated in Fig. 2 to investigate vision-based widget mapping approaches. To facilitate an effective evaluation of vision-based widget mapping approaches, we construct a real-world dataset and extract vision-based widget mapping approaches from the literature.

The dataset contains widget mapping relations, each of which involves a pair of semantically similar widgets from an app's iOS and Android versions. Each widget mapping relation provides inputs for the vision-based widget mapping approaches, including a source widget, a source screenshot of the app on the source platform displaying the source widget, and a target screenshot of the app on the target platform displaying a set of candidate widgets from which to identify the target widget. The mapping process in our study involves identifying the target widget from the candidate widgets, with

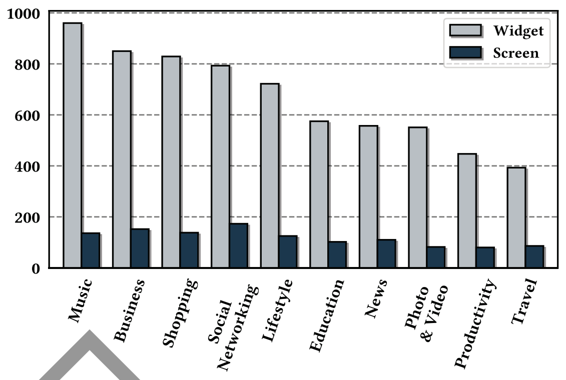


Fig. 3. Number of the collected widget mapping relations and the screen pairs displaying the mapped widgets of all app types in our real-world dataset.

one widget mapping relation providing the target widget to validate whether it has been located successfully.

B. A Real-world Dataset

To study vision-based widget mapping approaches in cross-platform GUI test migration, we need a real-world dataset. It is commonly believed that multiple platform version apps share consistency in e.g., widget graphics (e.g., images and icons), screen layout [16], and widget attributes [11], which supports test migration across platforms. However, previous research has shown that this belief is too crude and preliminary and needs to be improved. With this dataset, researchers can dig into the consistency and inconsistency in the visual aspect between the cross-platform versions of apps, and validate state-of-the-art techniques or design more efficient approaches. Unfortunately, we found that there is still a lack of such a real-world dataset to support further research.

To build our real-world dataset, we selected cross-platform apps that have both Android and iOS versions, which are widely used mobile platforms [11] and provide sufficient cross-platform apps for our study. Each unit of the dataset is a widget mapping relation, consisting of a pair of semantically similar widgets from the app's iOS and Android versions, along with screenshots of the app displaying these widgets and XML layout files. These XML layout files contain attributes for all widgets displayed in the screenshots, serving as locators for all widgets. We recorded a pair of mapped widgets by storing their locators together. Additionally, we recognized any remaining unmatched widgets using the attributes provided by the XML layout files.

To increase diversity, we selected 50 applications spanning ten different types from the Top Free App List in both the Apple Store and Google Play (Fig. 3). We opted for the latest versions of iOS apps, as the Apple Store only provides the latest version, to minimize version update differences when downloading Android apps from Google Play. We then executed both versions of the applications simultaneously on different devices. We manually explored and located mapped screens, and employed Appium [17] to automatically record corresponding screenshots and XML layout files at specific

TABLE I
ACTIVITY COVERAGE AND UNMATCHED WIDGETS. A IS ACTIVITIES, AC IS ACTIVITY COVERAGE, AND UMW IS UNMATCHED WIDGETS.

	# A	Avg. AC per App (%)	# UMW (Android)	# Avg. UMW per Screen (Android)	# UMW (iOS)	# Avg. UMW per Screen (iOS)
Business	32	10.44	2,432	16.21	4,376	29.17
Education	50	13.52	1,385	13.07	3,274	30.89
Lifestyle	33	10.16	1,992	15.56	4,036	31.53
Music	30	12.02	3,604	26.70	4,294	31.81
News	22	7.02	2,925	25.66	4,408	38.67
Photo & Video	20	11.93	1,693	20.15	2,720	32.38
Productivity	31	8.85	1,345	16.01	2,776	33.05
Shopping	84	12.54	3,199	22.53	4,117	28.99
Social Networking	49	6.31	1,937	11.46	5,736	33.94
Travel	44	14.60	1,482	17.44	3,213	37.80
Total	395	-	21,994	-	38,950	-

time intervals. Subsequently, we manually identified the semantically similar widgets across iOS and Android based on the mapped screens.

To ensure accuracy and avoid bias in the collection of widget mapping relations, a team of two Ph.D. candidates and two master students manually conducted the process. Due to the complexity and diversity of widget mapping relations, automated matching is still challenging. Therefore, the team relied on their observation and experience. The team was divided into two groups, each consisting of one Ph.D. candidate and one master student. The two groups independently collected mapped widgets from cross-platform apps, which were then cross-validated and merged as a single set of widget mapping relations. Conflicts were resolved by consensus among all team members. In case of disagreements, the two groups examined the actual apps together and made decisions jointly.

Dataset Overview. For a comprehensive overview of our dataset, Fig. 3 presents the number of widget mapping relations and the number of mapped screens in our dataset [13]. In its entirety, it includes 6,730 widget mapping relations from 1,197 pairs of screens across 50 popular cross-platform apps on iOS and Android (10 categories, 5 apps per category). Additionally, we report the activity coverage for each app category during the dataset construction, including the number of covered activities and the average activity coverage for each app within that category, and the total number of unmatched widgets and the average number of unmatched widgets per screen for both Android and iOS apps in our dataset. This information is presented in Table I.

C. Selected Techniques

Various vision-based widget mapping approaches use distinct descriptors that combine visual features and employ

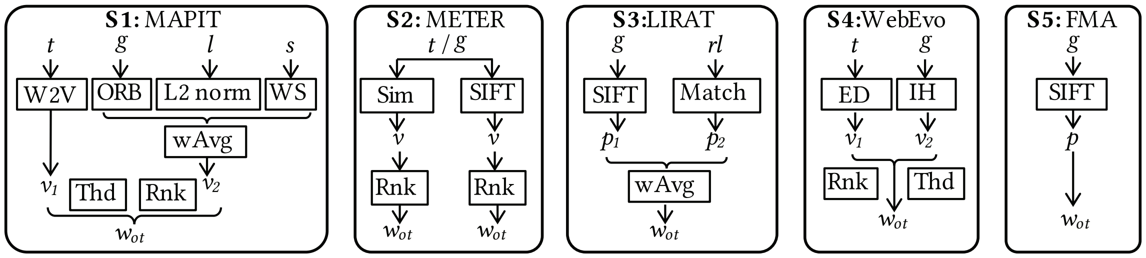


Fig. 4. Overview of the selected vision-based widget mapping approaches in this paper (S1-S5). t, g, l, s, and rl denote features *text*, *graphic*, *location*, and *relative location*. v is a similarity value, p is a coordinate, and w_{ot} is the output widget. W2V is Word2Vec, L2 norm is Euclidean distance [18], WS is widget size, wAvg is a weighted average formula, Sim is to calculate the similarity between two texts, Match is designed by LIRAT, ED is Levenshtein Edit Distance, IH is image hashing. Thd means adopting a similarity threshold. Rnk means ranking the similarity values.

diverse visual feature comparison algorithms and matching methods. These factors can have a significant impact on widget mapping. In our literature review, we found that only one technique, MAPIT [10], uses visual features for widget mapping in the context of cross-platform GUI test migration. However, it is insufficient for our study. We believe that our investigation should not be limited to cross-platform test migration but should also consider other GUI testing domains. Thus, we identified five state-of-the-art techniques and extracted their vision-based widget mapping approaches. Although GUI test migration across apps in the same category requires identifying semantically similar widgets between different apps, we excluded the existing techniques in this domain because they do not involve visual features or vision-based techniques. The selected techniques are as follows. Details are in Fig. 4.

- **S1.** MAPIT [10] is a cross-platform GUI test migration approach and currently considered state-of-the-art. We include its widget mapping approach and use the *text* information only from the screens.
- **S2.** METER [14] is a state-of-the-art approach for GUI test repair in case of changes in the GUI of an app. Using their techniques of detecting widget changes to measure the similarity between widgets, we can locate mapped widgets.
- **S3.** LIRAT [16] is a latest test record and replay technique. To replay a test on a new device, widget mapping techniques are necessary to locate the widget associated with the event to be triggered. We include this technique.
- **S4.** WebEvo [19] is one of the latest techniques to detect changes for evolving web pages. Similar as METER, We use WebEvo's ability to measure widgets' similarity in appearances and locate the mapped widgets across platforms.
- **S5.** Feature Matching-based Approach (FMA) [20] is recently provided to support visual GUI testing on Android. Visual GUI testing techniques require a widget mapping process that matches the screenshots of the widget with the event to be triggered in the test case to the actual widget on the screen. We use FMA's ability of locating mapped widgets via graphic comparisons.

The selected vision-based widget mapping approaches share similarities but differ in three key aspects: (1) the adopted visual feature combinations for descriptors, e.g., MAPIT uses four visual features, while FMA only uses feature *graphic*; (2) the feature comparison algorithms, e.g., MAPIT and METER both use feature *graphic*, but the selected feature comparison algorithms are different, i.e., ORB [21] and SIFT [22]; (3) the matching methods, e.g., MAPIT provides a threshold for *text*, while WebEvo provides a threshold for *graphic*.

D. Research Questions

We will answer the following research questions.

RQ1. ***Inconsistency Identification.*** *Are there any inconsistencies in the content of mapped widgets in their appearances between cross-platform versions of an app?*

We investigate whether there are inconsistencies in the appearance of widgets that may affect the effectiveness of vision-based widget mapping. To do this, we classify the widgets in our real-world dataset as either textual or graphical and identify any inconsistencies in the appearance of mapped widgets across platforms.

RQ2. ***Overall Effectiveness.*** *What is the most effective configuration to locate the semantically similar widgets between cross-platform versions of an app?*

We implement 89 configurations based on the selected approaches and evaluate all of them on the real-world dataset (Sec. III-B) collected by us in this paper to explore the vision-based widget mapping approaches in cross-platform GUI test migration and determine the most effective configuration.

RQ3. ***Internal Comparison.*** *What are the most effective instances of each step of a widget mapping approach?*

We study which instances perform the best in each step and recommend to prioritize these instances in future research.

E. Experiment Design and Settings

1) RQ1: Widgets convey functionality to users through the visual content in their appearances. The selected approaches are related with GUI testing and focus on the widgets' visual content in their appearances, which are then transferred as feature *graphic*. Some of them also transfer the content in the widget's appearance as feature *text* using OCR. Then, the selected approaches can adopt *graphic* or *text* or their combination to create descriptors aiming at supporting vision-based widget mapping. Different approaches employ different

	graphic comparison algorithm	*text* comparison algorithm	Variations (Feature combinations)	# Configurations
MAPIT	ORB, IH, SIFT-M	ED, Sim, W2V	(t, g, s, l), (t, g, s), (t, g, l), (t, s, l), (g, s, l), (t, g), (t, s), (g, s), (t, l), (g, l), (s, l), (l), (s), (g)	3*3*1*1+3*3*1+3*3*1+3*1*1+3*1*1 +3*3+3*1+3*1+3*1+3*1+1*1+1+1+3=**60**

	graphic	*text*	Variations	# Config.
METER	ORB, IH, SIFT-M	ED, Sim, W2V	(g, t)	3*3=**9**
WebEvo	ORB, IH, SIFT-M	ED, Sim, W2V	(g, t), (t)	3*3+3=**12**

	graphic	Variations	# Config.
LIRAT	ORB, IH, SIFT-M, SIFT-L, SIFT-F	(g, rl), (g), (rl), but some (g) are used in MAPIT	5*1+5*1+1-3=**8**
FMA	ORB, IH, SIFT-M, SIFT-L, SIFT-F	(g), but some (g) used in MAPIT and LIRAT	5-5=**0**

89

Fig. 5. Experiment configurations. *t*, *g*, *l*, *s*, and *rl* refer to *text*, *graphic*, *location*, *size*, and *relative location*. *size*, *location*, and *relative location* have one comparison algorithm. So, their choices are not listed.

algorithms to compare the feature *graphic* and *text* between the source widget and candidate widgets, e.g., SIFT and ORB for *graphic*. The mapping process is to find the widget pair with the highest similarity value.

Mapped widgets should have similar appearances since they represent the same app functionality on different platforms. However, since different teams develop cross-platform versions, inconsistencies in appearances can arise. These inconsistencies can impact vision-based widget mapping by affecting the performance of comparison algorithms. Therefore, inspired by METER, we classify the widgets in our real-world dataset as either textual or graphical. Then, we identify any inconsistencies in the textual and graphical contents of widgets' appearances across platforms to confirm their existence and enhance the evaluation of widget mapping.

We will manually classify all collected widget mapping relations. The team of four people (Sec. III-B) is divided into two groups as before, and both groups work on the entire dataset. They cross-validate and merge their results, and conflicts are resolved by all four people together.

2) RQ2 & RQ3: We conducted experiments with 89 different configurations for vision-based widget mapping, including the five originally selected approaches that serve as baselines for RQ2. These configurations are shown in Fig. 5. Each configuration was tested twice on our dataset, mapping iOS widgets to Android widgets and vice verse. The results were used to answer RQ2 and RQ3.

We identify three factors in Sec. II that can impact their ability of vision-based widget mapping: adopted visual feature combinations, feature comparison algorithms, and matching methods. Then, we set up the configurations by combining various choices of the three factors.

Visual feature combinations. As shown in Fig. 4, MAPIT uses *text*, *graphic*, *location*, and *size*; METER uses *text* and *graphic*; LIRAT uses *graphic* and *relative location*; WebEvo uses *text* and *graphic*; and FMA uses *graphic*. We generate variations of these approaches by using subsets of the adopted feature combinations. For example, LIRAT uses the combination of *graphic* and *relative location*, and we can create two variations by using only using *graphic* or *relative location* (for a total of $3 = 2^2 - 1$ configurations, including the original). In principle, MAPIT can provide 15 ($= 2^4 - 1$) configurations, METER can provide 3 ($= 2^2 - 1$), WebEvo can provide 3 ($= 2^2 - 1$), and FMA can provide 1 ($= 2^1 - 1$).

Feature comparison algorithms. We found 5 choices of feature comparison algorithms for feature *graphic*, 3 choices for *text*, and 1 choice for the others. The feature comparison algorithms for feature *graphic* are Scale Invariant Feature Transform (SIFT) [22], Oriented FAST and rotated BRIEF (ORB) [21], and Image Hashing (IH) [23]. As shown in Fig. 4, METER, LIRAT, and FMA use SIFT but with different supporting algorithms, e.g., FMA uses SIFT with RANSAC [24]. We denote them as SIFT-M, SIFT-L, and SIFT-F, respectively. The feature comparison algorithms for feature *text* are Word2Vec with tf-idf [25] (W2V for short), Edit Distance [26] (ED), and a customized formula (Sim). For feature *location*, Euclidean distance [18] (L2 norm) is used as the feature comparison algorithm. For feature *size*, the similarity values are determined by the difference in widget sizes normalized by the device size. The instance of feature *relative location* is proposed by LITRA [16]. LITRA divides the source and target screenshots into groups and assigns each widget a unique group number, row number, and column number within a grid. These numbers are combined to create a 3-tuple, representing the widget's *relative position*. The Match algorithm is then used to compare the similarity between these 3-tuples.

Matching methods. Each selected technique provides a distinct choice. MAPIT, METER, and WebEvo compute the similarity values of specified features between the source widget and all candidate widgets and generate similarity value lists for each feature. METER differentiates widgets based on their textual or graphical content and considers either *graphic* or *text* as a single active feature to construct a widget descriptor. Hence, METER ranks the comparison results of a single specified feature and outputs the widget with the highest similarity value in that feature. MAPIT and WebEvo rank multiple features and use a similarity threshold on one feature to determine an acceptable range of similarity values. Widgets whose comparison result of this feature is higher than the threshold can participate in further ranking. While MAPIT provides a threshold for *text*, WebEvo provides a threshold for *graphic*. Furthermore, MAPIT uses a weighted average formula to aggregate *graphic*, *location*, and *size* as a new similarity value. After filtering candidates by the threshold, WebEvo ranks them based on the similarity value of feature *text*, while MAPIT ranks them based on the new similarity values. LIRAT and FMA adopt different matching methods compared to the other approaches. Their feature comparison algorithms identify the most matching location within the target screenshots for a specified feature. FMA only adopts *graphic* as a descriptor and determines the output widget based on this location. LIRAT separately calculates the comparison results of *graphic* and *relative position* and outputs the two most matching locations for each feature. Then, it combines the coordinates of the two most matching locations using a customized weighted average formula to create a new coordinate for the final output widget.

As previously mentioned, MAPIT, METER, and WebEvo need similarity value lists generated by the feature comparison algorithms, and LIRAT and FMA need the most matching locations. While similarity value lists can be used to identify the most matching locations, the reverse is not true. So, SIFT-L and SIFT-F cannot be used by MAPIT, METER, and WebEvo.

The matching methods used by the selected approaches also restrict the acceptance of some subsets of adopted feature combinations. Specifically, WebEvo can not use *graphic* alone because its matching method uses a similarity value of *graphic* as a lower limit, which cannot determine the output widgets. Therefore, WebEvo can only adopt 2 kinds of feature combinations: *text* only or a combination of *graphic* and *text*. Similarly, MAPIT can not use *text* in isolation, so it can adopt 14 ($= 2^4 - 1 - 1$) kinds of feature combinations. METER uses both *text* and *graphic*, but only one of these features can be used as the descriptor of a widget in METER. Therefore, METER only adopts the one feature combination.

Remove redundant configurations. Fig. 5 presents all the possible choices for the key factors. By combining different instances of each factor, we can generate various configurations. Initially, there could be 97 different configurations of the selected techniques. However, some configurations are redundant. For example, LIRAT's variation that only adopts *graphic* using ORB, IH, and SIFT-M separately derives three configurations. They are covered by the configurations derived from MAPIT's variation that only adopts *graphic*. Similarly, all configurations derived from FMA are covered by the configurations derived from MAPIT's and LIRAT's variations which only adopt *graphic*. Hence, we finally have 89 different configurations in total.

3) Metrics: We aim to identify the target widget in a set of candidate widgets that is semantically similar to a given source widget, using various approaches. We evaluate the effectiveness of these approaches using mapped widgets across iOS and Android versions of an app from our real-world dataset. We use one of the mapped widgets as the source widget and the other as the target (i.e., the ground truth).

To access the performance of the selected approaches, we compute their precision, recall, and F1 values for each widget mapping request. A widget mapping request involves identifying the target widget which is semantically similar to a given source widget from the set of candidate widgets. These candidate widgets comprise all the widgets displayed in the target screenshots, irrespective of their involvement in any widget mapping relations. We calculate true positive (TP) when the source widget is correctly mapped to the target widget, false positive (FP) when the source widget is mapped to a wrong widget, and false negative (FN) when the source widget is not mapped to any widgets. Precision is computed as $TP/(TP+FP)$, *recall* is computed as $TP/(TP+FN)$, and F1 score is computed as $2 * prec * rec/(prec + rec)$.

TABLE II
DEFINITIONS OF THE CATEGORIES OF MAPPED WIDGETS.

Category		Definition
Textual	*T0*	The mapped widgets have the identical text in their appearances.
	T1	The mapped widgets have similar text in their appearances, but not identical.
	T2	The mapped widgets have text in their appearances that has no words in common.
Graphical	*G0*	The mapped widgets have the identical graphics in their appearances.
	G1	The mapped widgets have graphics with the same shape but different colors.
	G2	The mapped widgets have graphics with the same color but different shapes.
	G3	The mapped widgets have graphics with different shapes and colors.
Mixed	*Mix*	Of the mapped widgets, one has text while the other has a graphic in their appearances.

IV. Empirical Study Result

In the following, we report the results of our study.

A. RQ1: Inconsistency Identification

To address RQ1, we examine all widget mapping relations in our dataset (Sec. III-B) and classify them into three categories with eight subcategories, as shown in Table II. The three categories are textual mapped widgets (*T0-T2*), graphical mapped widgets (*G0-G3*), and mixed mapped widgets (*Mix*). The textual group is classified into three types based on shared words between source and target widgets. The graphical group is classified based on shape and color. Mixed mapped widgets occur when one widget contains text while the other has graphics. These categories are based on the consistency of content in the mapped widgets, which are extracted using the features *text* and/or *graphic* when mapping widgets. *T0* and *G0* describe the mapped widgets with consistent content in their appearances, while the others involve the mapped widgets with inconsistent content.

We present the classification results for all mapped widgets in our dataset in Fig. 6. The mapped widgets with consistent content (*T0* and *G0*) account for 71.9% of all the mapped widgets with fluctuations ranging from 59.5% (43.4% of *T0* plus 16.1% of *G0* for Social Networking apps) to 88.5% (57.3% of *T0* plus 31.2% of *G0* for Shopping apps), while mapped widgets with inconsistent content constitute the remaining 28.1%. *T0* outweighs *G0*, with both of them being the top two among all app types. *G1-G3* outnumber *T1-T2*. The distribution of *Mix* varies, ranging from 2.9% to 16.4%, across different app types.

The results show that over 71.9% of the mapped widgets have identical text and graphics, highlighting the potential for

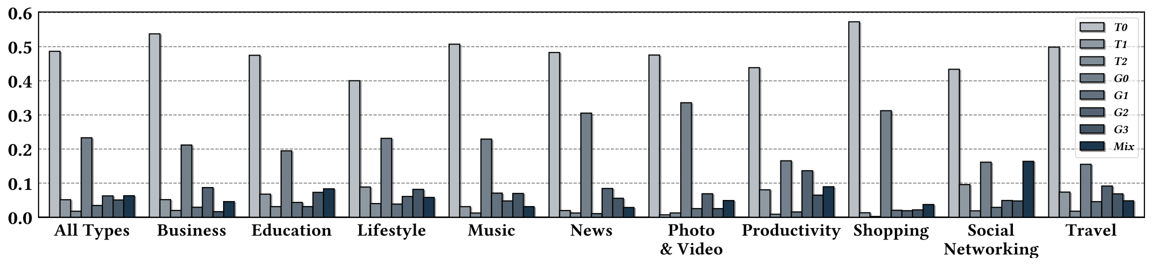


Fig. 6. The distributions of all categories of widget mapping relations in our dataset.

test reuse. Nonetheless, a notable proportion of mapped widget appearances display inconsistent content concerning the *text* and *graphic* features. This implies that relying solely on these features and their comparison algorithms, which check consistency between two widgets, to map widgets may not always be reliable, as the inconsistent content can negatively impact the widget mapping using vision-based methods. To address this issue, more powerful comparison algorithms for *text* and *graphic* are needed, and other visual features like *location* and *size* can be introduced to reduce the interference from inconsistency and enhance the mapping accuracy when combined with *text* and *graphic*. Introducing new features also requires well-designed matching methods to aggregate comparison results for each feature.

Finding: In our real-world dataset, approximately 28.1% of all mapped widgets display varying inconsistent content in their appearance, despite being semantically similar. This suggests a need for considering additional features and employing more sophisticated comparison methods.

B. RQ2: Overall Effectiveness

We implement 89 configurations, each of which handled 6,730 widget mapping requests for mapping iOS widgets to Android widgets and vice versa. Each widget mapping relation in our dataset yields two widget mapping requests. In each widget mapping request, one widget is designated as the source, and all widgets displayed on the target screen page are considered as potential candidates. Therefore, each configuration handles a total of 13,460 widget mapping requests.

The results obtained by using the 89 configurations for mapping 13,460 widgets from our dataset are illustrated in Fig. 7. Our evaluation of these configurations shows that the precision ranges from 11.61% to 81.57%, recall ranges from 17.41% to 94.75%, and F1 score ranges from 13.93% to 85.17%. The mean values of precision, recall, and F1 score of the 89 configurations are represented by the green dashed lines with circles at their endpoints on each box of Fig. 7, and they are 61.84%, 74.89%, and 66.75%, respectively. The median values of precision, recall, and F1 score are represented by the yellow solid lines on each box, and they are 65.93%, 81.59%, and 68.75%. When sorted by **precision**, the original methods (MAPIT, FMA, WebEvo, METER, and LIRAT) rank 41 (68.85%), 62 (57.57%), 77 (47.96%), 30 (70.92%), and 79 (41.70%), respectively. When sorted by **recall**, the rankings are 9 (93.74%), 30 (85.46%), 31 (84.22%), 69 (64.56%), and 88 (21.36%). Sorted by **F1 score**, the rankings are 24 (79.39%), 45 (68.79%), 64 (61.12%), 50 (67.59%), and 87 (28.25%). Out of the 89 configurations tested, only MAPIT appeared the in top 10% (top 9 entries) for recall. The precision of METER and MAPIT are higher than the mean and median values, while the recalls of MAPIT, FMA, and WebEvo are higher than the mean and median values. The F1 scores of MAPIT and FMA are higher than the mean value and median value, and METER's F1 score is higher than the mean value but lower than the median value. The others performed poorly in all metrics. Notably, almost all the configurations in the top 30% (top 27 entries) for precision, recall, and F1 score are derived from MAPIT, except for one configuration derived from LIRAT, which achieves the 11th place precision.

We selected the best configuration based on the F1 score, which uses IH for feature *graphic* and ED for feature *text*. It also uses feature *location* and *size* with their respective comparison algorithms, sets a threshold for the similarity value of *text*, and combines *graphic*, *location*, and *size* using a weighted average formula to generate a new similarity value. It finally ranks based on the combined similarity values, requiring *text* higher than the threshold. The best configuration ranks 5th in precision (78.18%) and 10th in recall (93.54%). Although the best values of precision and recall (81.57% and 94.75%) are achieved by different configurations, the best configuration selected is very close to them.

The best configuration provides improvements over the original approaches in F1 score, ranging from 5.78% for MAPIT, 16.38% for FMA, 17.58% for METER, 24.05% for WebEvo, and 56.92% for LIRAT. The best configuration achieves meaningful improvements for MAPIT, FMA, METER, and WebEvo. The significant improvement over LIRAT is due to its design not fitting iOS very well [16], which may worsen when mapping widgets across iOS and Android.

Finding: Numerous vision-based widget mapping configurations can be derived from existing techniques across various domains, with many surpassing the performance of current approaches. This indicates substantial potential for improvement in existing approaches.

C. RQ3: Internal Comparison

The selected approaches exhibit unique designs in each step of widget mapping, making it difficult to utilize some

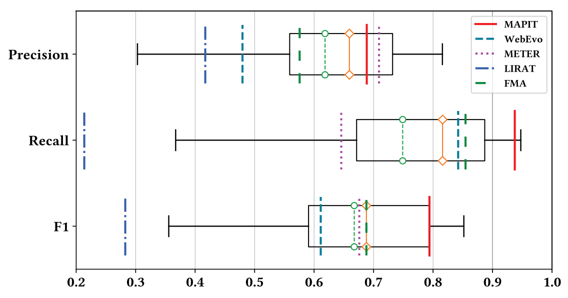


Fig. 7. All configurations' precision, recall, and F1 score values.

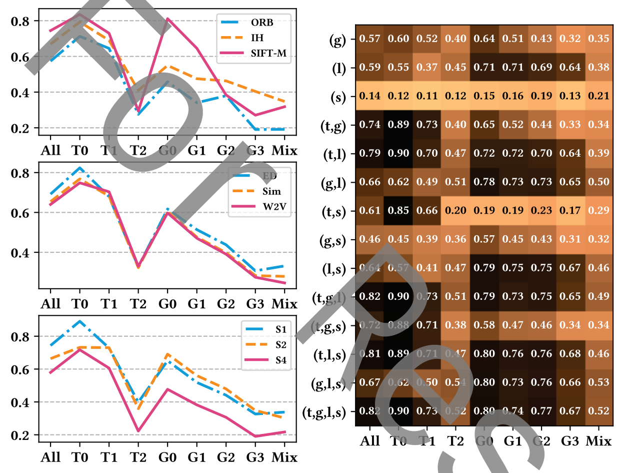


Fig. 8. The average F1 scores of the configurations using the same instances. The performance of comparison algorithms for feature *graphic* and *text* and matching methods are illustrated by three line charts on the left. On the right, a heat map is used to visualize the performance of different combinations of visual features. *t* refers to *text*, *g* refers to *graphic*, *l* refers to *location*, and *s* refers to *size*. The experiment results are presented for all mapped widgets and for different categories of mapped widgets.

of these designs in other approaches, which restricts the number of possible configurations. For example, there are 60 configurations derived from MAPIT, whereas only 9 are derived from METER. To ensure a fair analysis of the impact of the considered factors, we select two subsets of the 89 configurations for internal comparison. The first set of 27 configurations uses the combination of *text* and *graphic*. Within this set, we identify the most effective instances for the feature comparison algorithms (i.e., ED, Sim, and W2V for feature *text* and ORB, SIFT-M, and IH for *graphic*) and the matching method (i.e., S1-MAPIT, S2-METER, and S4-WebEvo). The second set includes all the configurations derived from MAPIT, involving feature *text* (t), *graphic* (g), *location* (l), and *size* (s). Based on the comparisons within this set, we will investigate the usage of visual features.

To compare the performances of different configurations in a specific aspect (e.g., the best instance of visual comparison algorithm for feature *graphic*), we calculate the average F1 scores of all configurations using the same instance of that aspect. This statistical approach helps to provide a fair evaluation of the instances' performances across the different configurations. The experiment results are depicted in Fig. 8. The left side of the figure contains three line charts displaying the results of the first configuration set, while the right side shows the results of the second configuration set in the form of a heat map. The figure showcases the performances of all the mapped widgets and the mapped widgets categorized into different types that were defined and identified in RQ1.

The **overall** experiment results (on all the mapped widgets) show that SIFT-M and ED are the best comparison algorithms for *text* and *graphic*, respectively, and the best matching method is S1 (MAPIT's method) for all the mapped widgets. The results also reveal that the feature combination *text*, *graphic*, *location*, and *size* (i.e., (t, g, l, s)) and (t, g, l) perform the best, while the combination (t, g, s) performs worse than (t, g), and (g, s) performs worse than (g). This suggests that adding *size* may negatively affect widget mapping. Additionally, using *location* and *size* together can result in better performance than using *location* without *size*, indicating that using *location* together with *size* can reduce the uncertainty of widget mapping caused by *size*. The **overall** internal comparison results are inconsistent with the best configuration found in RQ2, because the selected *graphic* feature comparison algorithm is different. Next, we will provide further analysis for **each kind of mapped widgets** within our classification.

Feature comparison algorithms. The results in the top line chart on the left of Fig. 8 indicate that: (1) SIFT-M and IH consistently outperform ORB in handling all kinds of mapped widgets; and (2) SIFT-M performs better than IH for *T0*, *T1*, *G0*, and *G1* mapped widgets, while IH performs better than SIFT-M for the other kinds. For graphical mapped widgets (*G0-G3*), since the feature *text* is empty, the performance is dominated by the comparison algorithms for *graphic*. SIFT-M significantly outperforms the other choices for *G0* and *G1* mapped widgets, which refer to identical mapped widgets and widgets with color inconsistencies, respectively. This suggests that SIFT-M is good at recognizing the consistent shape of graphics, ignoring differences in colors. IH considers both the shape and color of graphics, and hence performs better on *G2*, *G3*, and *Mix*, where the consistency of color is important. For textual mapped widgets (*T0-T2*), feature *graphic* is considered in most configurations. Similar text (*T0* and *T1*) can be treated as similar shapes if the textual content is considered as a kind of graphic. Thus, SIFT-M performs the best on *T0* and *T1*.

The middle line chart on the left of Fig. 8 reveals that the comparison algorithms for *text* (including ED, Sim, and W2V), exhibit very similar performances. In most cases, ED shows a slight advantage over the other two choices. This indicates that ED is the preferred choice for identifying mapped widgets, aligning with the overall results for all the mapped widgets.

Matching methods. The bottom line chart on the left of Fig. 8 shows that MAPIT's and METER's matching methods (S1 and S2) are close and significantly outperform WebEvo's matching method (S4) on all kinds of mapped widgets, except for mapped widgets with identical text (*T0*), where MAPIT performs the best. MAPIT and WebEvo set thresholds for *text* and *graphic* respectively, i.e., MAPIT uses *text* in a coarse granularity, while WebEvo uses *text* in high accuracy requirements. Further analysis shows that: (1) certain *T0* mapped widgets have multiple candidates in target screens displaying identical text; and (2) OCR may miss some characters [27], making the identical text not entirely consistent. These facts can affect the accuracy of using *text* to identify consistency, indicating the need to set a threshold for comparing text results. This conclusion aligns with the existing research [10].

We also observe that METER slightly outperforms MAPIT on graphical mapped widgets (*G0-G3*). This is because MAPIT uses feature *text* for mapping graphical widgets. When mapping graphical widgets, *text* is empty, which does not make contributions and dilute the reliability of identifying consistency between graphical mapped widgets when aggregating *graphic* and *text*. In contrast, METER distinguishes between textual and graphical content and only uses feature *graphic* for graphical widgets. Hence, using *text* in coarse granularity, e.g., a threshold, or not at all is recommended when mapping graphical widgets to avoid interference.

Visual feature combination. The heat map on the right of Fig. 8 shows that, in vision-based widget mapping for cross-platform test migration: (1) feature *location* can significantly improve the performance on graphical mapped widgets (*G0-G3*) and slightly improve the performance on *Mix* mapped widgets; (2) adding feature *size* to the original feature combination that includes *location* improves performance, but adding *size* alone does not improve performance; and (3) feature *text* can significantly improve the performance on textual mapped widgets (*T0-T2*). These observations lead to the conclusion that the most important visual features in vision-based widget mapping for cross-platform test migration are feature *location* and *text*, while the commonly-used *graphic* is less effective. This could be attributed to the fact that current feature comparison algorithms for *graphics* are able to identify consistencies but struggle to give measurable similarity values between the semantics of two widgets' graphics, particularly when there are inconsistencies. Additionally, feature *size* should be used together with *location*, as consistency identified via *size* may significantly overlap with that identified via *graphic* and/or *text*.

Findings: The design of vision-based widget mapping approaches require significant consideration, since:
(i) In terms of selecting visual features for semantic matching of widgets, more features do not always equate to better results; certain features must be used in conjunction with others to yield a positive impact; (ii) Features such as *text* are not advantageous for all types of widgets. The *text* feature should not be employed for mapped widgets displaying graphical content and should be applied only in a coarse-grained manner; (iii) Current comparison algorithms associated with the *graphic* feature struggle to effectively handle all types of graphical inconsistencies in widget appearances within our classification.

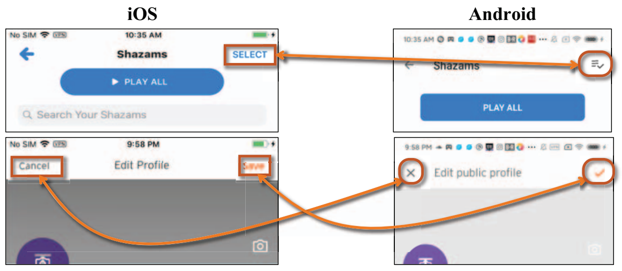


Fig. 9. Examples of *Mix* mapping relations in our dataset.

V. Discussion

Limitations. We focused on one-to-one widget mapping and carefully collected and verified one-to-one mapped widgets across platforms. However, one-to-many or many-to-one mapping can also occur, which poses challenges due to the indeterminacy of the number of widgets involved. Thus, our dataset might not fully assess the capabilities of the configurations of widget mapping approaches. To address this concern, we thoroughly examined the existing approaches under evaluation. It was found that the current approaches primarily focus on one-to-one mapping and rarely consider one-to-many or many-to-one mapping, often limited to simple scenarios. Hence, our evaluation could effectively reflect the current state of vision-based widget mapping approaches. We plan to investigate other types of widget mapping in future work. Furthermore, although the focus of this paper is on widget mapping, additional experiments are necessary to evaluate the impact of widget mapping on overall cross-platform GUI test migration. Nonetheless, widget mapping remains crucial and plays a significant role in test reuse.

Implication. In this paper, we explore vision-based widget mapping approaches for cross-platform GUI test migration. Our study results can facilitate GUI test migration across platforms with minimal effort. Moreover, our findings can be used to improve vision-based widget mapping and cross-platform GUI test migration approaches. Additionally, our study results on vision-based approaches can also be applied to other GUI testing domains, such as test record and replay [16], GUI test repair [14], [19], and visual GUI testing [20], as they can benefit from widget mapping. The study focuses on vision-based approaches, which have not been extensively explored. By combining our study results with existing techniques, we can improve not only test reuse but also other tasks related to GUI testing. This study identifies inconsistencies in mapped widgets' appearances to support our evaluation, which can guide further research in vision-based widget mapping. Machine learning techniques can be introduced to address these inconsistencies and improve widget mapping. For example, we identify mixed widget mapping relations (*Mix*), which contain widgets with graphics that are not comparable to text (Fig. 9). Image captioning techniques can be introduced to generate texts from graphics for comparison purposes.

Threats to validity. *External validity.* One potential threat to the external validity is the generalization of our results to other cross-platform apps or app scenarios. During the dataset construction, we did not achieve exhaustive activity coverage, resulting in potential oversights of certain widgets. To mitigate this, we invested considerable effort in collecting a diverse range of apps, selecting five apps from each of the ten different categories. This process ensures that our dataset collects a wide variety of widgets. Moreover, modern mobile apps often maintain consistent GUIs to offer seamless user experiences, resulting in visually similar widgets. By accumulating widgets from a diverse set of apps spanning different categories, this process can ensure greater diversity within our dataset. Although increasing activity coverage could enhance diversity, it requires extensive manual efforts to explore and identify new mapped screens, resulting in only marginal improvements. To balance efforts and efficiency, we followed the method outlined in this paper for dataset construction. *Internal validity.* A possible threat to internal validity is that there might be errors in our programs that led to wrong results. We migrated it by manually validating the correctness of feature comparison results and inspecting the screenshots of source and target widgets. If there is an abnormally high or low feature comparison results, we will check the source and all the candidate widgets with the screenshots displaying them. Additionally, we accounted for potential mistakes caused by human perception, such as discrepancies in color perception, by cross-validating the results with two independent groups. *Construct validity.* To minimize the potential threat to construct validity, we ensured that the selected approaches were faithfully re-implemented by referring to the original source code provided by the authors.

VI. Related Work

To the best of our knowledge, this paper presents the first empirical study on vision-based widget mapping for cross-platform GUI test migration. Recently, Zhao et al. [28] proposed the FrUITeR framework to evaluate test reuse techniques as a whole. It does not specifically focus on widget mapping. Mariani et al. [6] empirically studied events matching of test reuse techniques, but their work was limited to apps within the same category and only used textual information without visual features, focusing on Natural Language Processing (NLP) techniques for event mapping. In addition, there is currently no suitable dataset available to support widget mapping for cross-platform test migration. Therefore, our work collects such a dataset and investigates the application of visual features to support widget mapping for cross-platform GUI test migration.

There are currently two approaches for migrating tests across platforms: TestMig [11] and MAPIT [10]. TestMig requires the source code of both source and target apps and does not use any visual features to migrate tests. On the other hand, MAPIT uses many visual features and some textual information from the XML layout files. However, MAPIT may not be platform-agnostic for less well-known platforms, and its use of multiple visual features is preliminary and simple, without considering the consistency of these features or how to mitigate the impact of inconsistency. In our study, we include MAPIT. We also examine the latest vision-based widget mapping approaches used in other GUI testing domains, including: METER [14], which detects changes in app GUIs and repair GUI tests; WebEvo [19], which identifies changes in web page evolution that affect the GUI; LIRAT [16], which designs a vision-based approach to support test record and replay; and a feature matching-based approach [20], which enables visual GUI testing for Android. To migrate GUI tests across apps in the same category, it is necessary to identify semantically similar widgets between different apps. Previous studies by Behrang et al. [8], [29] and Lin et al. [9] constructed GUI models from the source code of apps to map widgets and reuse GUI tests. However, these approaches rely less on visual features or vision-based techniques. In contrast, our paper focuses on vision-based widget mapping approaches for cross-platform GUI test migration and excludes these techniques.

A survey [30] has been conducted on the use of computer vision (CV) algorithms in software engineering tasks. In our study, we utilize CV algorithms, e.g., SIFT, as visual feature comparison techniques for widget mapping in cross-platform test migration. Our study is the first to evaluate the effectiveness of CV algorithms in widget mapping for cross-platform test migration.

VII. Conclusion

This paper investigates the potential of reusing GUI tests across mobile app platform versions via vision-based widget mapping. We assess 89 configurations using a real-world dataset with 6,730 iOS and Android widget pairs, reporting crucial insights for improving the current design of the vision-based widget approaches.

Future research can explore promising avenues to address inconsistencies in vision-based widget mapping. One potential approach is to employ machine learning methods to precisely measure the similarity between the semantics of two widgets' graphics. This could enhance the accuracy of widget mapping in cases of visual inconsistencies. While this paper focused on one-to-one mapping, future studies can delve into one-to-many or many-to-one mapping techniques to enhance automatic GUI test migration techniques. Our findings can be leveraged to explore effective vision-based approaches for GUI test migration across platforms. Furthermore, it could serve as a guide for investigating vision-based techniques in other GUI testing domains, such as test record and replay, GUI test repair, and visual GUI testing. These explorations may yield valuable insights, thereby contributing to the advancement of GUI testing methods.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 62232014 and No. 61972193). We would like to express our gratitude to anonymous reviewers for their valuable comments.

REFERENCES

[1] K. Mao, M. Harman, and Y. Jia, "Sapienz: multi-objective automated testing for android applications," in *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*, A. Zeller and A. Roychoudhury, Eds. ACM, 2016, pp. 94–105. [Online]. Available: https://doi.org/10.1145/2931037.2931054

[2] M. Pan, A. Huang, G. Wang, T. Zhang, and X. Li, "Reinforcement learning based curiosity-driven testing of android applications," in *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*, S. Khurshid and C. S. Pasareanu, Eds. ACM, 2020, pp. 153–164. [Online]. Available: https://doi.org/10.1145/3395363.3397354

[3] M. Pan, Y. Lu, Y. Pei, T. Zhang, and X. Li, "Preference-wise testing of android apps via test amplification," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 1, feb 2023. [Online]. Available: https://doi.org/10.1145/3511804

[4] M. L. Vásquez, C. Bernal-Cárdenas, K. Moran, and D. Poshyvanyk, "How do developers test android applications?" in *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*. IEEE Computer Society, 2017, pp. 613–622. [Online]. Available: https://doi.org/10.1109/ICSME.2017.47

[5] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for android: Are we there yet? (E)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, M. B. Cohen, L. Grunske, and M. Whalen, Eds. IEEE Computer Society, 2015, pp. 429–440. [Online]. Available: https://doi.org/10.1109/ASE.2015.89

[6] L. Mariani, A. Mohebbi, M. Pezzè, and V. Terragni, "Semantic matching of GUI events for test reuse: are we there yet?" in *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, C. Cadar and X. Zhang, Eds. ACM, 2021, pp. 177–190. [Online]. Available: https://doi.org/10.1145/3460319.3464827

[7] F. Behrang and A. Orso, "Test migration for efficient large-scale assessment of mobile app coding assignments," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018*, F. Tip and E. Bodden, Eds. ACM, 2018, pp. 164–175. [Online]. Available: https://doi.org/10.1145/3213846.3213854

[8] ——, "Test migration between mobile apps with similar functionality," in *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 2019, pp. 54–65. [Online]. Available: https://doi.org/10.1109/ASE.2019.00016

[9] J. Lin, R. Jabbarvand, and S. Malek, "Test transfer across mobile apps through semantic mapping," in *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 2019, pp. 42–53. [Online]. Available: https://doi.org/10.1109/ASE.2019.00015

[10] S. Talebipour, Y. Zhao, L. Dojcilovic, C. Li, and N. Medvidovic, "UI test migration across mobile platforms," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 756–767. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678643

[11] X. Qin, H. Zhong, and X. Wang, "Testmig: migrating GUI test cases from ios to android," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*, D. Zhang and A. Møller, Eds. ACM, 2019, pp. 284–295. [Online]. Available: https://doi.org/10.1145/3293882.3330575

[12] J. Qian, Y. Ma, C. Lin, and L. Chen, "Accelerating ocr-based widget localization for test automation of GUI applications," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 2022, pp. 6:1–6:13. [Online]. Available: https://doi.org/10.1145/3551349.3556966

[13] "Real-world data set," 2023. [Online]. Available: https://github.com/RuihuaJi/vision-based-widget-mapping

[14] M. Pan, T. Xu, Y. Pei, Z. Li, T. Zhang, and X. Li, "Gui-guided test script repair for mobile apps," *IEEE Trans. Software Eng.*, vol. 48, no. 3, pp. 910–929, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2020.3007664

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, Eds., 2013, pp. 3111–3119.

[16] S. Yu, C. Fang, Y. Yun, and Y. Feng, "Layout and image recognition driving cross-platform automated mobile testing," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 1561–1571. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00139

[17] "Appium," 2021. [Online]. Available: https://appium.io/

[18] "Euclidean distance," 2021. [Online]. Available: https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095800175

[19] F. Shao, R. Xu, W. A. Haque, J. Xu, Y. Zhang, W. Yang, Y. Ye, and X. Xiao, "Webevo: taming web application evolution via detecting semantic structure changes," in *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*. ACM, 2021, pp. 16–28. [Online]. Available: https://doi.org/10.1145/3460319.3464800

[20] L. Ardito, A. Bottino, R. Coppola, F. Lamberti, F. Manigrasso, L. Morra, and M. Torchiano, "Feature matching-based approaches to improve the robustness of android visual GUI testing," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 2, pp. 21:1–21:32, 2022. [Online]. Available: https://doi.org/10.1145/3477427

[21] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, D. N. Metaxas, L. Quan, A. Sanfeliu, and L. V. Gool, Eds. IEEE Computer Society, 2011, pp. 2564–2571. [Online]. Available: https://doi.org/10.1109/ICCV.2011.6126544

[22] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision, Kerkyra, Corfu, Greece, September 20-25, 1999*. IEEE Computer Society, 1999, pp. 1150–1157. [Online]. Available: https://doi.org/10.1109/ICCV.1999.790410

[23] W. Jiang, G. Er, Q. Dai, and J. Gu, "Similarity-based online feature selection in content-based image retrieval," *IEEE Trans. Image Process.*, vol. 15, no. 3, pp. 702–712, 2006. [Online]. Available: https://doi.org/10.1109/TIP.2005.863105

[24] D. Barath and J. Matas, "Graph-cut RANSAC," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 6733–6741. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Barath_Graph-Cut_RANSAC_CVPR_2018_paper.html

[25] A. N. Aizawa, "An information-theoretic perspective of tf-idf measures," *Inf. Process. Manag.*, vol. 39, no. 1, pp. 45–65, 2003. [Online]. Available: https://doi.org/10.1016/S0306-4573(02)00021-3

[26] E. S. Ristad and P. N. Yianilos, "Learning string-edit distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 5, pp. 522–532, 1998. [Online]. Available: https://doi.org/10.1109/34.682181

[27] X. Xiao, X. Wang, Z. Cao, H. Wang, and P. Gao, "Iconintent: automatic identification of sensitive UI widgets based on icon classification for android apps," in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 257–268. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00041

[28] Y. Zhao, J. Chen, A. Sejfia, M. S. Laser, J. Zhang, F. Sarro, M. Harman, and N. Medvidovic, "Fruiter: a framework for evaluating UI test reuse," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 1190–1201. [Online]. Available: https://doi.org/10.1145/3368089.3409708

[29] F. Behrang and A. Orso, "Test migration for efficient large-scale assessment of mobile app coding assignments," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018*, F. Tip and E. Bodden, Eds. ACM, 2018, pp. 164–175. [Online]. Available: https://doi.org/10.1145/3213846.3213854

[30] M. Bajammal, A. Stocco, D. Mazinanian, and A. Mesbah, "A survey on the use of computer vision to improve software engineering tasks," *IEEE Trans. Software Eng.*, vol. 48, no. 5, pp. 1722–1742, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2020.3032986