2023 IEEE 24th International Conference on Information Reuse and Integration for Data Science (IRI) | 979-8-3503-3458-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/IRI58017.2023.00057

"What You See Is What You Test": Recommending Features from GUIs for Requirements-Based Testing

Zedong Peng
University of Cincinnati
Cincinnati, OH, USA
pengzd@mail.uc.edu

Juha Savolainen
Danfoss Drives A/S
Graåten, Denmark
juha.savolainen@danfoss.com

Jianzhang Zhang
Hangzhou Normal University
Hangzhou, China
jianzhang.zhang@foxmail.com

Nan Niu
University of Cincinnati
Cincinnati, OH, USA
nan.niu@uc.edu

***Abstract*—Requirements-based testing (RBT) advocates the design of test cases in order to adequately exercise the behavior of a software system without regard to the internal details of the implementation. To address the challenge that requirements descriptions may be inaccurate in practice, we align requirements engineering and software testing in a novel way by not counting on a complete and up-to-date requirements documentation. Rather, we maintain the black-box nature of RBT to recommend features as the units of testing from software's graphical user interfaces (GUIs). In particular, we exploit optical character recognition (OCR) to identify the textual information from GUIs, and further build the GUI-feature correspondences based on software's user-centric documentation which may exhibit partial correctness. Such correspondences from multiple software systems in the same domain serve as a foundation for our recommendation engine, which suggests the to-be-tested features related to a given GUI. We demonstrate our recommender's feasibility with a study of five products in the web conferencing domain, and the results show the more complete set of features against which a GUI needs to be tested.**

***Index Terms*—graphical user interfaces, requirements engineering and testing, recommender, feature testing**

I. Introduction

Requirements-based testing (RBT) is a process of ensuring that the requirements are of high quality, and then designing test cases to adequately cover the requirements without regard to the internal details of the software implementation [1], [2]. Skoković and Skoković [3] further claim that, once the tests generated from all the requirements execute successfully against the code, 100% of the functionality has been verified and the code is ready to be delivered into production.

In practice, however, requirements specifications rapidly fall out of date when they exist at all [4]. Indeed, software evolution is a major reason for the requirements document to become out of sync [5], especially in high-speed agile development where new and changing features are continuously deployed in short cycles [6].

A *feature*, as defined by Zave [7], is an increment of functionality usually with a coherent purpose. Cisco, for instance, released an average of 14 features monthly in 2022, and this pace continued in 2023 [8]. In version 42.2 released in February 2022, the changing features include: "raise hand button moved to control bar", and the improved features include: "real-time transcription for 13 spoken languages" as well as "allowing host and cohost to change registration form". This last feature is illustrated in Figure 1 where both the textual description and the graphical user interface (GUI) depiction are used in the official release site. Compared to the textual description which is often brief, the GUI provides more details about the structure of how developers implement the feature and how end users interact with it.

GUI testing is therefore an important activity for checking the user-facing implementation's correctness and for revealing failures (mainly, system crashes [9]) before the end users encounter them. Current techniques typically interact with the available widgets by following the structure of the GUI, without any consideration about the functions that are executed [10]. To address the challenge, we present a framework to recommend the features against which a GUI needs to be tested. The GUI shown in Figure 1, for example, is relevant not only to the "allowing host and cohost to change registration form" feature, but also to "preassigning participants to breakout sessions" and maybe other features. Thus, what we address in this paper is to identify the requirements knowledge in the context of RBT for interactive applications, i.e., applications that the users interact with through GUIs [10].

Our framework exploits product-level GUI-feature mappings so as to establish the mapping data from multiple sources (products) of the same domain. These sources then support our recommender via federated search [11], aiming to retrieve from the cumulative mapping data at the domain level a set of features in response to a GUI query. To process GUIs in our framework, we leverage state-of-the-art optical character recognition (OCR) tools, and further investigate the matching of GUIs by means of their OCR-ed texts. We show our framework's applicability and effectiveness through a study of five web conferencing software systems, illuminating the more accurate features relevant to a GUI.

The main contribution of our work is the automated recommendation of which requirements-level features to test for a GUI. Such knowledge provides support for RBT even when a software system's requirements are not documented completely or updated continually, which we believe offers a practical solution to requirements engineers and testers. In what follows, we present background information in Section II. Section III details our framework, Section IV describes the empirical evaluation, and finally, Section V concludes the paper.

979-8-3503-3458-6/23/$31.00 ©2023 IEEE
DOI 10.1109/IRI58017.2023.00057

Allow host and cohost to change registration form

Hosts and cohosts can change the registration form – registrant number, registration ID, and approval rules option even after someone has registered for the webinar.

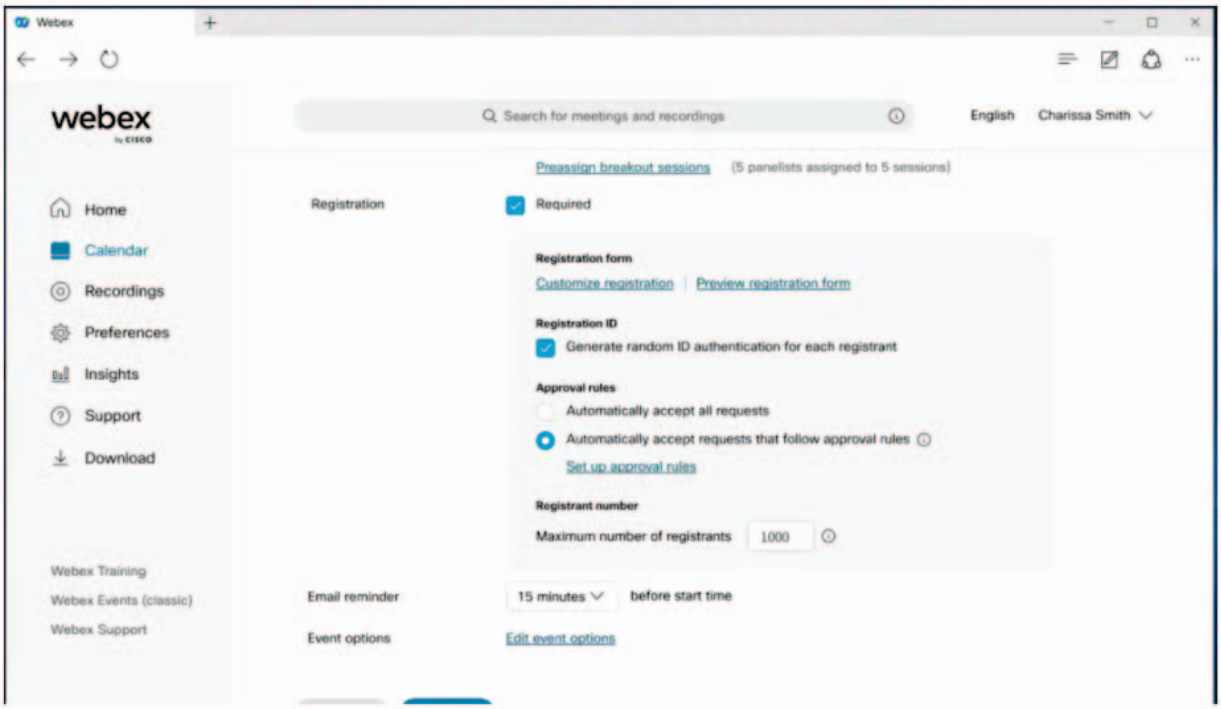


Fig. 1. Illustration of a newly released feature of Webex where textual and GUI (graphical user interface) information is presented [8].

II. Background and Related Work

RBT can be regarded as a key part of the broader effort to align requirements engineering (RE) and software testing where black-box testing is more suited than white-box testing. Whittaker [12] provided a system testing example for a text editor with the following regular expression:

$$\text{Filemenu.Open filename}^{*}\ (\text{ClickOpen} \mid \text{ClickCancel}) \quad (1)$$

where the asterisk represents the Kleene closure operator indicating that the filename action can occur zero or more times. The expression of (1) indicates that the first input received is Filemenu.Open followed by zero or more selections of a filename (with a combination of mouse clicks and keyboard entries), and then either the Open or Cancel button is pressed. This simple yet implicit model represents every combination of inputs that can happen to test the file-open task, whether they make sense or not [12].

To make the model of system testing more explicit, Hsia *et al.* [13] introduced scenario analysis supporting requirements analysts' manual construction of a finite state machine from the user's view. An equivalent regular grammar could then be generated, from which the tests could be derived. Our recent work exploits natural language processing techniques to infer tests from feature descriptions [14], [15].

Despite the growing literature on RBT, a practical challenge is the availability of high-quality requirements information documented in a fast-evolving software project. For example, the study by Bjarnason *et al.* [16] shows that, in the software development unit of a company producing network equipment, there are no documented requirements for around half of the incoming issue reports. We are therefore interested in exploiting the interrelatedness of software artifacts (especially features and GUIs) to better support the RE-testing alignment.

A GUI test consists of: (i) a sequence of events that interact with the GUI, and (ii) one or more assertion oracles that predict on the GUI state [17]. Using the GUI shown in Figure 1 as an example, we can define a two-event test input: selecting "30 minutes" in the "Email reminder" option, followed by clicking the "Set up approval rules" link. Hence, the structure of widgets is significant for GUI testing.

The study by Choudhary *et al.* [18] shows that, even though there exist elaborated techniques that use complex structural information, the most effective testing technique for Android applications is still a technique that simply performs random clicks on the GUI. Another limitation of structure-

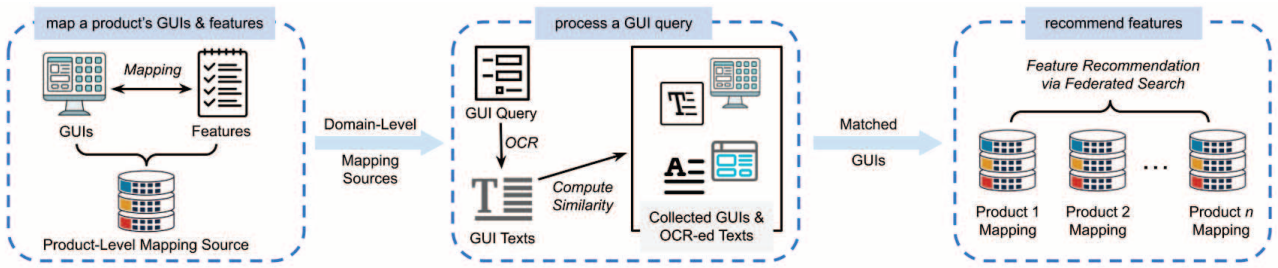


Fig. 2. Components of the framework for recommending a set of features against which a GUI is tested.

based explorative GUI testing is the use of system crashes as implicit oracles, without asserting oracles about functional correctness [19], [20].

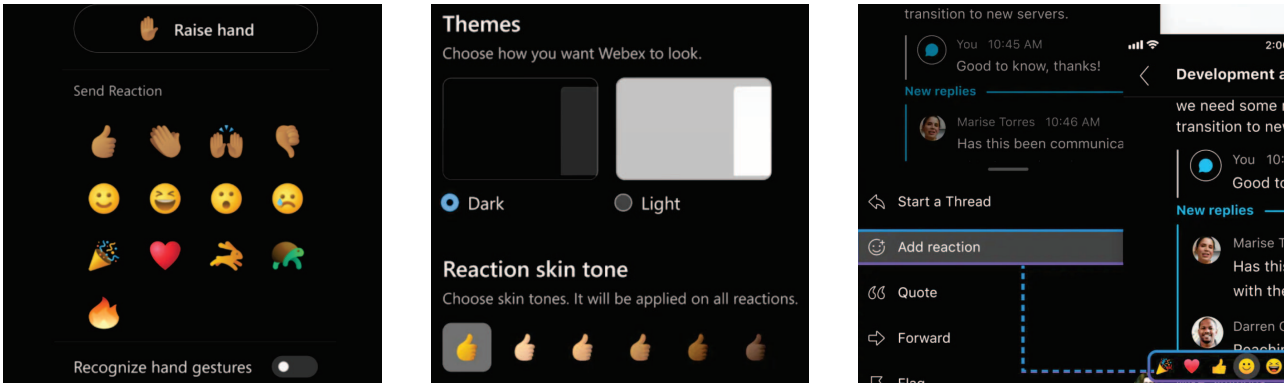
Recently, Mariani *et al.* [10] performed structural matching of GUIs and application-independent operations like authentication, thereby equipping test cases with functional oracles. Inspired by the generation of GUI tests that stimulate semantically relevant usage scenarios [10], we aim to identify application-specific functionalities involving a GUI. Yet, we posit the GUI reuse across similar applications [17], [19], [21], [22] holds the key to finding the relevant features for functional testing. Next we present a framework for recommending the features associated with a GUI over a domain of software systems that share common functionalities.

III. Finding Features to Test a GUI

The main research challenge that we address in this paper is: "Given a GUI, what features need to be tested?" The overview of our solution is presented in Figure 2. Three major steps are involved in our framework, which we elaborate in this section. Recall that our work applies to GUI-rich interactive applications, and hence we use contemporary web conferencing software to illustrate our framework in this section.

Mapping GUIs and Features. We use a product's authoritative sources to identify the GUI-feature mappings. For Cisco's Webex, the data sources include the product's release notes, its help center, and its official blog site. We refer to these data sources as *user-centric documentation*, since the primary goal is to inform the end users of the product. Due to this goal, GUIs are frequently included to effectively communicate the usage and effect of a feature.

A few GUIs and features of Webex, together with their data sources, are provided in Figures 3a and 3b respectively. Although textual descriptions of a feature exist, they are brief in most cases. For each of the five features listed in Figure 3b, the description contains one to three sentences. Figure 3c shows the mappings that we are able to establish between GUIs and features according to Webex's authoritative user-centric documentation. We currently perform our framework's



(a) Sample graphical user interfaces of Webex [8] (from left to right): GUI_1, GUI_2, and GUI_3

ID	Title
F_1	Use reactions in your meetings
F_2	Choose the skin tone for your reactions
F_3	Choose the color theme
F_4	Add emoji reactions to messages
F_5	New reactions and improved raise hand notifications

(b) Sample features of Webex

Mapping Established	
GUI_1	$\{F_1, F_2, F_5\}$
GUI_2	$\{F_1, F_2, F_4, F_5\}$
GUI_3	$\{F_4\}$

(c) GUI-feature mappings of Webex

Fig. 3. Illustrations of Webex GUIs, features, and their mappings.

first step manually, though automated scraping may increase the efficiency of data extraction.

Matching GUIs via Texts. The second step of our framework is concerned with matching a query GUI with the GUIs already collected at the end of the first step. Although GUIs can be compared directly in their image formats (e.g., through the underlying pixel values), the textual information contained in the GUI is crucial for software engineering tasks, such as inferring user-facing source code from mobile apps' screenshots [23], and generating GUI tests [24]. In fact, both of the previous studies [23], [24] leveraged optical character recognition (OCR) tools in order to extract texts from images.

Building on the prior work, we match GUIs in our framework via their textual representations via the state-of-the-art CRAFT method [25] and the Tesseract engine [26]. For example, the terms recognized from the GUI of Figure 1 include: "webex", "Home", "Calendar", etc. Once the texts of a GUI are recognized by our OCR tool chain, we cast the GUI matching problem to the textual matching problem. We further investigate two algorithms in the second step of our framework: Jaccard similarity coefficient and Kendall rank correlation coefficient [27].

In our matching of GUIs via their OCR-ed texts, Jaccard similarity is concerned with the content, whereas Kendall's τ considers both the content and the layout. To accommodate the texts' close matching (e.g., "event" $\approx$ "events") we adopt fuzzy matching with the Levenshtein distance at the term level [28] and set the matching threshold to be 0.8. When there exist multiple matches meeting the threshold, the match with the longest common subsequence is kept.

Retrieving Features through Federated Search. As the GUI-feature mappings established for an individual product at the first step of our framework exhibit partial correctness, the feature recommendation at the last step of Figure 2 takes advantage of *federated search* [11] to compensate for the product-level incompleteness. Federated search responds to a query by retrieving the relevant information from multiple data sources at once.

Of particular importance to federated search is how to blend the search results from various data sources so that *one* integrated set of results is returned to user's query. Two blending techniques can be distinguished here: search merging and data fusion [11]. In both cases, GUI ranking and feature consolidation are required. The difference lies in the ordering of these operations. Search merging blends the top-ranked results returned for the query by different sources into a single list. On the contrary, data fusion consolidates the features first vertically within a data source. The consolidated features are then ranked by the GUIs to produce the final result list.

To highlight the difference of the two blending techniques, we refer to the hypothetical data presented in Table I. Let us assume the GUI matching threshold is 0.6 and above. If search merging is applied, the single list of ranked GUIs will be GUI_γ, GUI_δ, and GUI_α. Further performing feature consolidation from this list will lead to the final recommended features: $\{\text{f}_1\,(3),\ \text{f}_2\,(1),\ \text{f}_3\,(1)\}$ where the number in the parentheses denotes a feature's tally. If the blending is done by data fusion, then feature consolidation may first produce "0.55 as the matching score, $\text{GUI}_{\alpha\beta}$: $\{\text{f}_1\,(1),\ \text{f}_2\,(2),\ \text{f}_3\,(1)\}$" for Product 1, and "0.75 as the matching score, $\text{GUI}_{\gamma\delta}$: $\{\text{f}_1\,(2),\ \text{f}_3\,(1)\}$" for Product 2. The final result of data fusion after applying the $\geq$0.6 GUI matching threshold will then be: $\{\text{f}_1\,(2),\ \text{f}_3\,(1)\}$. As can be seen from this hypothetical example, the features recommended by search merging are different from those recommended by data fusion. We present the evaluation of real-world products next.

TABLE I
HYPOTHETICAL DATA FROM TWO SOURCES

Matching Score	Product 1
0.6	GUI_α: $\{\text{f}_1, \text{f}_2\}$
0.5	GUI_β: $\{\text{f}_2, \text{f}_3\}$

Matching Score	Product 2
0.8	GUI_γ: $\{\text{f}_1\}$
0.7	GUI_δ: $\{\text{f}_1, \text{f}_3\}$

IV. EMPIRICAL EVALUATION

A. Research Questions

We set out to investigate three research questions (RQs) to assess our framework.

RQ_1: How accurate are the features recommended for a query GUI?

The overall performance of our framework rests on the quality of the features recommended for a given GUI. We measure recommendation quality in RQ_1 by precision, i.e., the extent to which the recommended features are truly something that the GUI needs to be tested against.

RQ_2: Do the recommended features improve the completeness of a GUI's RBT?

The practical value of our framework can be demonstrated if the recommendations enrich the functional aspects of testing a GUI. We thus answer RQ_2 by comparing the recommendation results with the manually identified features from a product's authoritative documentation.

RQ_3: To what extent does the domain knowledge contribute to the feature recommendation?

If searching the product's sole data source makes little difference than searching through multiple products, then not much value is added by federated search. Our RQ_3 therefore examines how much help, if at all, various sources provide to feature recommendation.

B. Subject Systems

We chose to study five products in the web conferencing domain (i.e., Webex, Zoom, Teams, VooV, and DingTalk), due to our own domain familiarity and personal experience with these software applications. Another reason is the availability of the product's authoritative user-centric documentation from which we extracted GUIs, features, and their mappings. For

each product, we manually collected 20+ GUIs and between 21 and 44 features.

C. Results and Analysis

To answer $\mathbf{RQ}_1$, we randomly selected 10 GUIs from each product to serve as the queries. We then executed our recommendation framework to generate a ranked list of features in response to each GUI query. Because GUI matching could be done by Jaccard or Kendall's τ, and federated search could be carried out by search merging or data fusion, we explored all the four implementations of our framework. Figure 4 plots the average precision values across the 50 GUI queries. Our comparisons focus on the top-5 and top-10 recommended features, and hence Figure 4 visualizes precision at 5 (P@5) and precision at 10 (P@10).

Not surprisingly, P@10 is less than P@5 no matter which implementation is adopted. This implies more irrelevant features (false positives) are included when a longer recommendation list is kept. A rather surprising result is that the Jaccard, set-of-terms matches (Figures 4c and 4d) outperform their Kendall's τ, sequence-of-terms counterparts (Figures 4a and 4b). The result indicates that similar GUIs may match well in their textual contents, but requiring the matching contents to also follow a consistent layout can be a tall order. Clearly, a "male, female" GUI corresponds to a "female, male" GUI; however, their Kendall's τ score would be the lowest.

While relevant features have been recommended for a GUI, $\mathbf{RQ}_2$ examines the recommendation completeness from a practical angle, i.e., if the top-recommended features add to what is already known from the product's own user-centric documentation, then the to-be-tested features established by our framework can lead to a more complete coverage of RBT than what would otherwise be done by following the existing documentation. To that end, we answer $\mathbf{RQ}_2$ qualitatively through an example.

In Figure 5a, one of VooV's GUI queries is shown. VooV's authoritative documentation maps GUI_{18} of Figure 5a with two features: {"Screen sharing", "Whiteboard sharing"}. Our framework returns VooV's GUI_{20} of Figure 5b as one of the top-ranked candidates, which further leads to a consolidated "Screen watermark sharing" feature being recommended. Adding watermark helps to protect the content in the shared space, e.g., it discourages non-owner's screen printing due to the presence of the watermark. The watermark feature, thus, is relevant to VooV's GUI_{18} from functional testing's standpoint. A tester, equipped with our recommendations, would ensure "Screen watermark sharing" behaves correctly when GUI_{18} is invoked, effectively enriching the scope of the functionalities for the GUI under testing. For the 10 queries that we experimented on Webex, Zoom, Teams, VooV, and DingTalk, 2, 2, 3, 4, and 5 GUIs received additional feature recommendations beyond the products' current documentation, achieving the completeness improvement on 32% of the GUIs when considering the top-5 recommended features.

In illustrating the previous research question, we used both GUIs of Figure 5 from the same product. To gain insights into how much help there is in the domain, we answer $\mathbf{RQ}_3$ by distinguishing the relevant features recommended from a product's own source and those recommended from the other products' sources. Table II reports $\mathbf{RQ}_3$ results when the top-5 feature recommendations are taken into account. The counts in Table II represent *unique* features: If a feature is consolidated from a product's own source and the other products' sources, then that feature is counted only once and the count is placed in the middle column of Table II. It is encouraging to observe from the rightmost column of Table II that about one to three relevant features have their origins completely outside a product's own boundary, showing the complementary effect [29] of federated search.

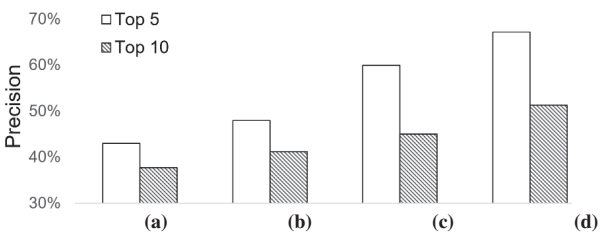
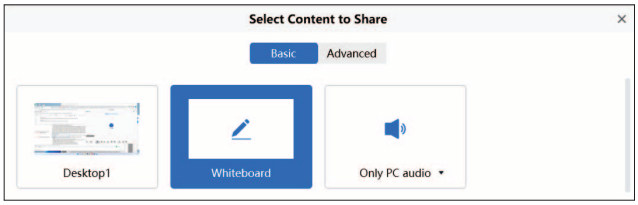


Fig. 4. P@5 and P@10 comparisons: **(a)** Kendall's τ + data fusion, **(b)** Kendall's τ + search merging, **(c)** Jaccard + data fusion, and **(d)** Jaccard + search merging.



(a) VooV GUI_{18}

(b) VooV GUI_{20}

Fig. 5. Query **(a)** and the top-ranked GUI result **(b)**.

TABLE II
NUMBER OF UNIQUE AND RELEVANT FEATURES FROM DIFFERENT SOURCES (AVERAGE OVER 10 GUI QUERIES PER PRODUCT)

Product	Product's own data source	Four other products' data sources
Webex	2.3	0.8
Zoom	1.9	0.9
Teams	1.9	1.5
VooV	1.0	2.8
DingTalk	1.0	2.5

V. CONCLUSIONS

Software testing with requirements knowledge can be effective in reducing project failures, defects, and rework [3]. To overcome the practical difficulties of maintaining accurate and updated requirements-to-test information, we present a novel framework to recommend features that form a foundation for a GUI's functional testing.

Our work can be extended in several avenues. Empirical studies with more software systems and more application domains are needed to lend strength to the findings reported here. We are also interested in experimenting GUI matching via bag-of-terms models, as well as image-based deep learning methods. Moreover, we plan to develop systematic and automated ways of feature consolidation in order to improve our current manual work on the task.

ACKNOWLEDGMENT

This work is partially support by Engineering Research Center of Mobile Health Management System of Chinese Ministry of Education (Grant No. 2022GCZXGH04).

REFERENCES

[1] T. Bhowmik, S. R. Chekuri, A. Q. Do, W. Wang, and N. Niu, "The role of environment assertions in requirements-based testing," in *Proceedings of the International Requirements Engineering Conference (RE)*, Jeju Island, South Korea, September 2019, pp. 75–85.

[2] Z. Peng, P. Rathod, N. Niu, T. Bhowmik, H. Liu, L. Shi, and Z. Jin, "Environment-driven abstraction identification for requirements-based testing," in *Proceedings of the International Requirements Engineering Conference (RE)*, Notre Dame, IN, USA, September 2021, pp. 245–256.

[3] P. Skoković and M. Rakić-Skoković, "Requirements-based testing process in practice," *International Journal of Industrial Engineering and Management*, vol. 1, no. 4, pp. 155–161, 2010.

[4] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, May 2015.

[5] T. Gorschek and M. Svahnberg, "Requirements experience in practice: Studies of six companies," in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds. Springer, 2005, pp. 405–424.

[6] N. Niu, S. Brinkkemper, X. Franch, J. Partanen, and J. Savolainen, "Requirements engineering and continuous deployment," *IEEE Software*, vol. 35, no. 2, pp. 86–90, March/April 2018.

[7] P. Zave, "FAQ sheet on feature interaction," Last accessed: June 13, 2023. [Online]. Available: http://www.pamelazave.com/faq.html

[8] Cisco, "What's New for the Latest Channel of Webex Meetings," last accessed on June 13, 2023. [Online]. Available: https://help.webex.com/en-US/article/xcwws1/What's-New-for-the-Latest-Channel-of-Webex-Meetings

[9] A. M. Memon, I. Banerjee, and A. Nagarajan, "What test oracle should I use for effective GUI testing?" in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, Montréal, Canada, October 2003, pp. 164–173.

[10] L. Mariani, M. Pezzè, and D. Zuddas, "Augusto: exploiting popular functionalities for the generation of semantic GUI tests with oracles," in *Proceedings of International Conference on Software Engineering (ICSE)*, Gothenburg, Sweden, May-June 2018, pp. 280–290.

[11] M. Shokouhi and L. Si, "Federated search," *Foundations and Trends in Information Retrieval*, vol. 5, no. 1, pp. 1–102, 2011.

[12] J. A. Whittaker, "What is software testing? And why is it so hard?" *IEEE Software*, vol. 17, no. 1, pp. 70–79, January/February 2000.

[13] P. Hsia, J. Samuel, J. Gao, D. C. Kung, Y. Toyoshima, and C. Chen, "Formal approach to scenario analysis," *IEEE Software*, vol. 11, no. 2, pp. 33–41, March 1994.

[14] S. Sturmer, N. Niu, T. Bhowmik, and J. Savolainen, "Eliciting environmental opposites for requirements-based testing," in *Proceedings of the International Requirements Engineering Conference (RE) Workshops*, Melbourne, Australia, August 2022, pp. 10–13.

[15] Z. Peng, P. Rathod, N. Niu, T. Bhowmik, H. Liu, L. Shi, and Z. Jin, "Testing software's changing features with environment-driven abstraction identification," *Requirements Engineering*, vol. 27, no. 4, pp. 405–427, December 2022.

[16] E. Bjarnason, M. Unterkalmsteiner, M. Borg, and E. Engström, "A multi-case study of agile requirements engineering and the use of test cases as requirements," *Information & Software Technology*, vol. 77, pp. 61–79, September 2016.

[17] L. Mariani, A. Mohebbi, M. Pezzè, and V. Terragni, "Semantic matching of GUI events for test reuse: Are we there yet?" in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, Held Virtually, July 2021, pp. 177–190.

[18] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for Android: Are we there yet?" in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, Lincoln, NE, USA, November 2015, pp. 429–440.

[19] J.-W. Lin, R. Jabbarvand, and S. Malek, "Test transfer across mobile apps through semantic mapping," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA, November 2019, pp. 42–53.

[20] Y. Zhao, T. Yu, T. Su, Y. Liu, W. Zheng, J. Zhang, and W. G. J. Halfond, "ReCDroid: Automatically reproducing Android application crashes from bug reports," in *Proceedings of International Conference on Software Engineering (ICSE)*, Montréal, Canada, May 2019, pp. 128–139.

[21] F. Behrang and A. Orso, "Test migration between mobile apps with similar functionality," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA, November 2019, pp. 54–65.

[22] A. Rau, J. Hotzkow, and A. Zeller, "Transferring tests across web applications," in *Proceedings of International Conference on Web Engineering (ICWE)*, Cáceres, Spain, June 2018, pp. 50–64.

[23] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with REMAUI," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, Lincoln, NE, USA, November 2015, pp. 248–259.

[24] S. Yu, C. Fang, Y. Yun, and Y. Feng, "Layout and image recognition driving cross-platform automated mobile testing," in *Proceedings of International Conference on Software Engineering (ICSE)*, Madrid, Spain, May 2002, pp. 1561–1571.

[25] Y. Baek, B. Lee, D. Han, S. Yun, and H. Lee, "Character region awareness for text detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA, June 2019, pp. 9365–9374.

[26] R. Smith, "An overview of the Tesseract OCR engine," in *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, Curitiba, Brazil, September 2007, pp. 629–633.

[27] Z. Peng, X. Lin, N. Niu, and O. I. Abdul-Aziz, "I/O associations in scientific software: A study of SWMM," in *Proceedings of the International Conference on Computational Science (ICCS)*, Krakow, Poland, June 2021, pp. 375–389.

[28] X. Lin, M. Simon, and N. Niu, "Scientific software testing goes serverless: Creating and invoking metamorphic functions," *IEEE Software*, vol. 38, no. 1, pp. 61–67, January/February 2021.

[29] W. Wang, N. Niu, M. Alenazi, J. Savolainen, Z. Niu, J.-R. C. Cheng, and L. D. Xu, "Complementarity in requirements tracing," *IEEE Transactions on Cybernetics*, vol. 50, no. 4, pp. 1395–1404, April 2020.