Engineering Accessible Software

Arun Krishnavajjala
George Mason University
akrishn@gmu.edu

Kevin Moran
University of Central Florida
kpmoran@ucf.edu

***Abstract*—This paper discusses a research area at the intersection of Machine Learning, Software Engineering, and Human-Computer Interaction research to create intelligent developer tools for more accessible mobile applications. Our research goals revolve around combining software engineering and accessibility research to create developer facing tools that facilitate the development of accessible software. We introduce MOTOREASE, an innovative approach that utilizes computer vision and text processing to identify accessibility issues in mobile app UIs for *motor-impaired users*. The tool detects four motor-impaired user-focused UI design guidelines: t ouch target size, expanding sections, persisting elements, and adjacent icon distance, with an average accuracy of around 90%. This represents a significant step towards improving software accessibility for all users.**

I. INTRODUCTION

Software plays a crucial role in our lives, impacting everything from web and smartphone applications to desktop systems. The increasing dependence on critical software applications, such as banking and communication tools, highlights the need for accessibility. People with diverse abilities often face challenges in using software as it is currently designed. The World Health Organization reports that 15% of people have some disability, making software accessibility even more important [1].

Efforts to improve accessibility have been driven not only by ethical motivations but also by government policies. The United States Government, through the American with Disabilities Act (ADA) [2], requires public websites and services to be accessible. Despite progress, research has shown that there is still an abundance of accessibility issues in applications [4, 5, 9, 16–18, 22, 28, 30, 33]. Existing accessibility guidelines and tools are often overlooked or difficult for developers to utilize effectively.

Beyond providing equitable access to software for users with a variety of backgrounds, accessibility features often improve user experience more broadly, as many accessibility guidelines are designed following the general principals of universal design [3], in that the adherence to such guidelines is more likely to lead to an improved user experience for *all* users [29].

Software engineering research is constantly innovating how software is made. Software testing and developer tools are constantly evolving and are making their way into the accessibility space. Software testing has been around for decades, but has recently grown into a more complex field with the use of computer vision and machine learning techniques to automatically generate and run tests.

Our research leverages leading advancements in machine learning, software engineering, human-computer interaction (HCI), and accessibility research. It aims to create intelligent developer tools that can integrate accessibility into the software development process. Current accessibility research often focuses on creating solutions tailored to specific disabilities, with emphasis on low vision users. Research on Deaf and Hard of Hearing (DHH) and motor-impaired users is comparatively limited [6, 8, 10, 11, 21, 23, 25–27, 32, 34].

HCI research has been instrumental in understanding how users with various abilities interact with their devices and assistive tools. Software engineering research has also made strides in testing frameworks and developer tools for accessibility. Automated testing using computer vision and machine learning is becoming more prevalent [7, 13, 24, 26]. Intelligent tools can help developers identify and incorporate accessibility features efficiently.

By bridging HCI, accessibility, and software engineering research, we can create exciting opportunities to develop accessible software. Leveraging data and intelligent tools, we can address accessibility challenges during the developmental and design stages, making software more inclusive for all users.

II. RESEARCH GOALS

Given the abundance of inaccessible applications and the lack of developer tools to address them, we have explored a different approach, based on the creation of intelligent tools that will help that will help developers efficiently improve the accessibility of their applications. This goal is achieved by combining the visual aspects of the screen along with the underlying functionality of the screen to assist developers in their pursuit to make more accessible applications.

A. Advancing Inclusive Software Development

One of the primary research goals is to advance the field of inclusive software development, specifically focusing on improving accessibility in software design and implementation. The research will entail creating and enhancing software development tools to empower developers to build applications with accessibility features since the early stages of development. We aim to investigate novel gesture recognition techniques, UI element modifications, and provision of accessible datasets to support developers in catering to diverse user needs. Furthermore, we seek to automate methods for enhancing user

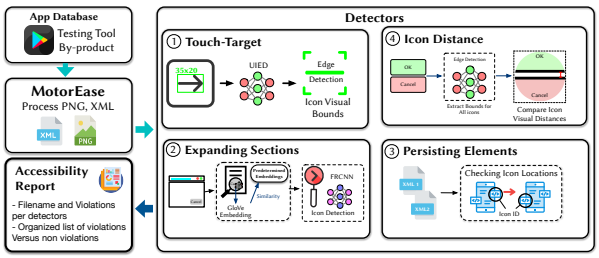

Fig. 1. Overview of MOTOREASE's Workflow

interfaces with accessibility features and labels to facilitate better user experiences for individuals with disabilities.

B. Automated Accessibility Testing and UI Understanding

Another crucial research goal is to develop automated tools for accessibility testing and comprehensive understanding of user interfaces. Automating accessibility-focused application testing will be instrumental in promptly identifying violations of accessibility guidelines in software. Our research efforts will focus on creating innovative testing frameworks that can assess accessibility measures and features, particularly for Android and iOS applications. Additionally, we aim to explore automated visual understanding of user interfaces, especially those found in smartphone applications, to develop algorithms that can label elements and propose design changes to accommodate the needs of individuals with various accessibility requirements.

By improving accessibility in design and implementation, we strive to make technology more usable for individuals with disabilities. Through innovative software development tools and automated testing frameworks, we aim to streamline the creation of accessible applications. Our research on automated visual understanding of user interfaces seeks to create smarter, more intuitive software for diverse accessibility needs. These contributions represent steps towards a more inclusive technological landscape. One current project that reflects these goals is presented in Section 3.

III. THE MOTOREASE APPROACH

In this section, we present MOTOREASE, an implemented and tested automated tool to detect motor-impairment accessibility issues in mobile apps. MOTOREASE is an automated approach that aims to detect motor-impairment accessibility guideline violations by analyzing UI metadata and screenshots collected via AIG tools. MotorEase operates in three stages, and implements four guideline violation detectors, as depicted in Figure 1. First, an AIG tool is run on a target application to produce a set of screenshot and `uiautomator XML` files (*i.e.*, UI metadata) before and after each AIG tool action. We tailor our approach to utilize UI metadata generated using the `uiautomator` framework, which captures UI layout information in a structured `XML` format, as this is most prevalent uility used by recent Android AIG tools [12, 14, 15, 19, 20, 31]. Second, MOTOREASE utilizes a series of four *violation detectors* to analyze the screenshots and UI metadata to determine if the target application failed to follow motor-impairment guidelines. Finally, MOTOREASE collects the information from the detectors and compiles an *accessibility report* that informs developers of accessibility guideline violations.

TABLE I
OVERALL RESULTS FOR EACH DETECTOR

Metrics	Touch-Target	Exp. Section	Pers. Elements	Icon Dist
Precision	1.0000	0.9042	0.8214	0.7119
F1-Score	0.7986	0.9129	0.8846	0.8317
Accuracy	0.8525	0.9123	0.8776	0.9575
Sensitivity	0.6648	0.9205	0.9583	1.0000
Recall	1.0000	0.9042	0.8000	0 9525

Developers are in need of a tool which will address the four accessibility issues detected by MOTOREASE. The goal is to provide developers with a reliable tool to facilitate the development of accessible applications. To achieve our study goals, we formulated the following five research questions to evaluate our approach:

- **RQ**$_1$ *How accurate is the Expanding Section detector?*
- **RQ**$_2$ *How accurate is the Visual Touch Target detector?*
- **RQ**$_3$ *How accurate is the Persisting Element detector?*
- **RQ**$_4$ *How accurate is the UI Element Distance detector?*
- **RQ**$_5$ *Does* MOTOREASE *identify a limited number of false positive and negative violations?*

The first four research questions (RQ$_1$ - RQ$_4$) focus on evaluating the detection quality of each of MOTOREASE's detectors. It is important for each detector to be accurate and provide the developer with accurate accessibility violation predictions. To achieve this, we use the evaluation metrics of accuracy, precision, sensitivity, and recall. This evaluation presents results from Table I, showing accuracy, precision, sensitivity, and recall of the detectors for true positive (TP) and true negative (TN) values. The final research question RQ$_5$ focus on evaluating the tool's ability to accurately predict violations. The importance of TP and TN values can be crucial to developers to properly identify accessibility violations within their applications.

A. RQ$_1$: Expanding Section Detector Accuracy

The expanding section detector performed well across nearly all of our studied metrics, as indicated in Table I. With a precision of 0.9042, F1-Score of 0.9129, and an accuracy of 0.9123, this detector shows promising results that it is capable of identifying a section's "collapsibility" accurately. This indicates that, by using MOTOREASE, developers will have an increased chance of identifying sections that were designed without a method of closure, and hence may impeded use by motor-impaired users. However, this detector does struggle to detect certain instances in the MOTORCHECK benchmark.

B. RQ$_2$: Visual Touch Target Detector Accuracy

The touch-target detector showcased strong performance, as depicted in Table I. With impeccable precision, an F1-Score of 0.7986, and an accuracy of 0.8525, the detector demonstrates promising capabilities in effectively recognizing and categorizing screens with touch target issues. These outcomes underscore its proficiency. The detector effectively equips developers with valuable insights into diminutive icons

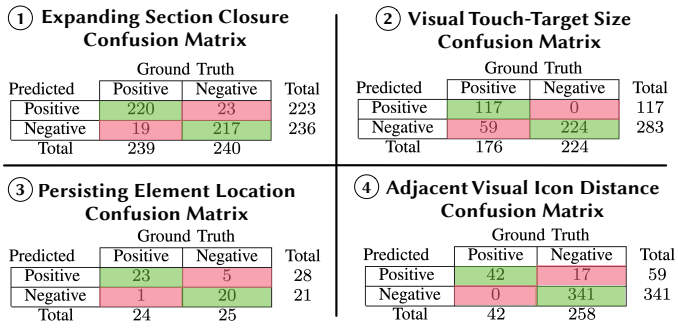

Fig. 2. Detector Confusion Matrices

that might inconvenience users with tremors and imprecise touches.

Nonetheless, although this detector predominantly delivers accurate results, it does exhibit certain limitations in its detection scope. Notably, it fails to identify specific types of violations. For instance, it is unable to spot icons on the screen that lack the "clickable" label in the XML code. In cases involving dynamically generated screenshots, comprehensive metadata for each screen element might not be consistently available. This lack of information adversely affects the detector's capacity to extract interactive elements on a screen. The inability to identify clickable elements stands as a central reason behind inaccuracies or instances of overlooked violations.

C. RQ_3: Persisting Element Detector Accuracy

The outcomes of this detector are showcased in Table I. With a precision of 0.8214, an F1-Score of 0.8846, and an accuracy rating of 0.8786, this detector shows promising potential. Furthermore, the detector's recall rate of 0.9583 indicates its proficiency in accurately identifying true positives. However, it's important to note that this detector heavily depends on the XML structure to pinpoint screen elements, potentially resulting in instances of mis-classification

D. RQ_4: Visual Icon Distance Detector Accuracy

The findings of this detector are shown in Table I. With a precision of 0.7119, an F1-Score of 0.8317, and an accuracy level of 0.9575, this detector's performance stands out. With a flawless recall rate, it becomes evident that developers are equipped with a dependable tool capable of precisely identifying closely positioned icons. This capability prompts potential adjustments in UI design and icon placement, enhancing overall user experience.

Similar to the Visual Touch-Target Violation detector, this particular detector heavily relies on the metadata provided by `uiautomator` to identify clickable components, along with the accuracy of the UIED element bound detector. The latter contributes to certain instances of inaccurately reported violations, primarily attributed to inaccuracies in overlapping boundary specifications.

E. RQ_5: False Positives and Negatives

The confusion matrices for the detectors are shown in Figure 2, where green boxes illustrate predictions that matched the ground truth, and red boxes illustrate predictions that did not match the ground-truth. These figures provide a visual representation of false positive and negative rates. Figure 2-① illustrates that the visual touch target detector never produced a false-positive outcome. This is due to the nature of the detector. The detector specifically extracts elements labeled as clickable in the `XML`, therefore once they are extracted and have their edges analyzed, the detector is able to detect violations with certainty. The confusion matrix for the expanding section detector is shown in Figure 2-③, there were 23 false positive predictions, mainly due to limitations related to lexical pattern matching. Finally, MOTOREASE's persisting element detector identified only 8 false positives, mainly due to inconsistencies in matching elements across screens due to unexpected changes in `uiautomator XML` files. In evaluating the effectiveness of MOTOREASE, we also considered the impact of false negative predictions, as they signal violations that are not flagged by MotorEase, and hence could reach end-users.

In summary, MotorEase exhibits effective detection of motor-impairment accessibility guideline violations, particularly in popular open-source Android applications. The results show promise for enhancing accessibility testing in app development, and the tool's accuracy and applicability make it a valuable addition to the field of accessibility evaluation for Android apps.

IV. FUTURE WORK

In this paper we presented MOTOREASE, an approach for detecting, classifying, reporting motor-impairment accessibility violations. We measured the performance, generalizability, and applicability of MOTOREASE to various open source applications. Our results indicate that MOTOREASE is effective in practice and offers a novel approach for developers to identify accessibility issues affecting motor-impaired users.

MOTOREASE exemplifies the culmination of our research goals, aimed at empowering developers in their pursuit of software accessibility. By equipping developers with the efficient means to detect motor-impairment accessibility issues within their applications, MotorEase marks a step forward in creating a more inclusive and accessible digital landscape.

Building on the success of MotorEase, our future projects envision the creation of a search engine tailored to the needs of developers. This novel tool will enable developers to input prototype screens, allowing them to find analogous, more accessible design alternatives. Moreover, the search engine will analyze the provided screens, thereby identifying potential accessibility concerns that warrant attention and resolution. This approach empowers developers to proactively address accessibility considerations, even before commencing the coding phase. This approach is currently in development and will be tested in the near future.

Our overarching vision revolves around promoting accessibility inclusivity throughout the software development lifecycle. By providing developers with the necessary tools and resources, we strive to usher in a new era of software design

and development that embraces accessibility as a core tenet, ensuring that software products are accessible to users of diverse abilities and backgrounds. Through sustained research efforts and diligent technological advancements, we aim to contribute significantly to the realization of an accessible and inclusive digital realm for all.

REFERENCES

[1] World report on disability http://www.who.int/disabilities/world_report/2011/en/, 2011.

[2] Adalaws https://www.ada.gov/cguide.htm, 2019.

[3] https://www.section508.gov/blog/universal-design-what-is-it/, Universal Design Definition and Guidelines.

[4] A. Aizpurua, M. Arrue, S. Harper, and M. Vigo. Are users the gold standard for accessibility evaluation? In *Proceedings of the 11th Web for All Conference*, W4A '14, New York, NY, USA, 2014. Association for Computing Machinery.

[5] L. D. A. Almeida and M. C. C. Baranauskas. Universal design principles combined with web accessibility guidelines: A case study. In *Proceedings of the IX Symposium on Human Factors in Computing Systems*, IHC '10, page 169–178, Porto Alegre, BRA, 2010. Brazilian Computer Society.

[6] M. Bajammal and A. Mesbah. Semantic web accessibility testing via hierarchical visual analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1610–1621, 2021.

[7] M. Bajammal and A. Mesbah. Semantic web accessibility testing via hierarchical visual analysis. In *Proceedings of the 43rd International Conference on Software Engineering*, ICSE '21, page 1610–1621. IEEE Press, 2021.

[8] G. Brajnik, C. Pighin, and S. Fabbro. Model-based automated accessibility testing. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '15. Association for Computing Machinery, 2015.

[9] H. N. da Silva, A. T. Endo, M. M. Eler, S. R. Vergilio, and V. H. S. Durelli. On the relation between code elements and accessibility issues in android apps. In *Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing*, SAST 20, page 40–49, New York, NY, USA, 2020. Association for Computing Machinery.

[10] M. M. Eler, J. M. Rojas, Y. Ge, and G. Fraser. Automated accessibility testing of mobile apps. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 116–126, 2018.

[11] K. Z. Gajos, J. O. Wobbrock, and D. S. Weld. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, page 231–240, New York, NY, USA, 2007. Association for Computing Machinery.

[12] T. Gu, C. Sun, X. Ma, C. Cao, C. Xu, Y. Yao, Q. Zhang, J. Lu, and Z. Su. Practical gui testing of android applications via model abstraction and refinement. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, page 269–280. IEEE Press, 2019.

[13] J. Li, Z. Yan, E. H. Jarjue, A. Shetty, and H. Peng. Tangiblegrid: Tangible web layout design for blind users. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022. Association for Computing Machinery.

[14] Y. Li, Z. Yang, Y. Guo, and X. Chen. Droidbot: a lightweight ui-guided test input generator for android. In *ICSE-C*. IEEE, 2017.

[15] K. Mao, M. Harman, and Y. Jia. Sapienz: Multi-objective automated testing for android applications. In *ISSTA*. ACM, 2016.

[16] D. A. Mateus, C. A. Silva, M. M. Eler, and A. P. Freire. Accessibility of mobile applications: Evaluation by users with visual impairment and by automated tools. In *Proceedings of the 19th Brazilian Symposium on Human Factors in Computing Systems*, IHC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[17] T. B. McHugh and C. Barth. Assistive technology design as a computer science learning experience. In *Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '20, New York, NY, USA, 2020. Association for Computing Machinery.

[18] K. Montague, H. Nicolau, and V. L. Hanson. Motor-impaired touchscreen interactions in the wild. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers ; Accessibility*, ASSETS '14, page 123–130, New York, NY, USA, 2014. Association for Computing Machinery.

[19] K. Moran, M. Linares-Vasquez, C. Bernal-Cardenas, C. Vendome, and D. Poshyvanyk. Crashscope: A practical tool for automated testing of android applications. 2018.

[20] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk. Automatically discovering, reporting and reproducing android application crashes. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 33–44, 2016.

[21] K. Norman, Y. Arber, and R. Kuber. How accessible is the process of web interface design? In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '13, New York, NY, USA, 2013. Association for Computing Machinery.

[22] U. Oh, S. K. Kane, and L. Findlater. Follow that sound: Using sonification and corrective verbal feedback to teach touchscreen gestures. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '13, New York, NY, USA, 2013. Association for Computing Machinery.

[23] K. Park, T. Goh, and H.-J. So. Toward accessible mobile application design: Developing mobile application accessibility guidelines for people with visual impairment. In *Proceedings of HCI Korea*, HCIK '15, page 31–38, Seoul, KOR, 2014. Hanbit Media, Inc.

[24] Y.-H. Peng, M.-T. Lin, Y. Chen, T. Chen, P. S. Ku, P. Taele, C. G. Lim, and M. Y. Chen. Personaltouch: Improving touchscreen usability by personalizing accessibility settings based on individual user's touchscreen interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–11, New York, NY, USA, 2019. Association for Computing Machinery.

[25] N. P. K. Ramachandra and C. Csallner. Testing web-based applications with the ¡u¿v¡/u¿oice ¡u¿c¡/u¿ontrolled ¡u¿a¡/u¿ccessibility and ¡u¿t¡/u¿esting tool (vcat). In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ICSE '18, page 208–209, New York, NY, USA, 2018. Association for Computing Machinery.

[26] N. Salehnamadi, A. Alshayban, J.-W. Lin, I. Ahmed, S. Branham, and S. Malek. Latte: Use-case and assistive-service driven automated accessibility testing framework for android. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[27] N. Salehnamadi, F. Mehralian, and S. Malek. Groundhog: An automated accessibility crawler for mobile apps. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE 2022)*, 2022.

[28] M. T. Santiago and A. B. Marques. Are user reviews useful for identifying accessibility issues that autistic users face? an exploratory study. In *Proceedings of the 21st Brazilian Symposium on Human Factors in Computing Systems*, IHC '22, New York, NY, USA, 2022. Association for Computing Machinery.

[29] Z. Sarsenbayeva, N. van Berkel, E. Velloso, J. Goncalves, and V. Kostakos. Methodological standards in accessibility research on motor impairments: A survey. *ACM Comput. Surv.*, may 2022. Just Accepted.

[30] G. M. S. Silva, R. M. de C. Andrade, and T. de Gois R. Darin. Design and evaluation of mobile applications for people with visual impairments: A compilation of usable accessibility guidelines. In *Proceedings of the 18th Brazilian Symposium on Human Factors in Computing Systems*, IHC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[31] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su. Guided, stochastic model-based gui testing of android apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, page 245–256, New York, NY, USA, 2017. Association for Computing Machinery.

[32] C. Vendome, D. Solano, S. Liñán, and M. Linares-Vásquez. Can everyone use my app? an empirical study on accessibility in android apps. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 41–52, 2019.

[33] S. Yan and P. G. Ramachandran. The current status of accessibility in mobile apps. *ACM Trans. Access. Comput.*, 12(1), feb 2019.

[34] X. Zhang, L. de Greef, A. Swearngin, S. White, K. Murray, L. Yu, Q. Shan, J. Nichols, J. Wu, C. Fleizach, A. Everitt, and J. P. Bigham. Screen recognition: Creating accessibility metadata for mobile applications from pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.