**UX Journey Based Automation Application Design: Visual Interface and LLM to Improve Multilevel User Accessibility in Windows**

M. Amin Hidayatullah1, a) and Evi Dwi Wahyuni2, b)

Author Affiliations

1 Informatics Engineering, Universitas Muhammadiyah Malang, Malang, Indonesia

2 Department of Computer Science, Universitas Muhammadiyah,Malang, Indonesia.

Author Emails  
a) Corresponding author: m\_amin@webmail.umm.ac.id  
b)anotherauthor: [evidwi@umm.ac.id](mailto:evidwi@umm.ac.id)

**Abstract.** Task automation in Windows environments using tools like AutoIt, AutoHotkey, and PowerShell still faces significant accessibility challenges for non-technical users due to reliance on manual scripting skills and complex technical documentation. This study aims to design an intuitive GUI-based automation application interface to bridge accessibility gaps for multi-level users (technical and non-technical) in Windows. The proposed solution integrates three key approaches: (1) a drag-and-drop interface for building automation workflows without code, (2) live action recording (macro recording) of user interactions, and (3) Large Language Models (LLMs) like ChatGPT and DeepSeek as conceptual script-generating assistants.The development methodology adopts the iterative UX Journey framework (Discover, Explore, Test, Listen) through mixed-methods design. The Discover phase leverages LLMs for SWOT analysis, literature review, and mapping user behavioral variables. The Explore phase involves co-design sessions with 5 multi-level participants to develop personas, user scenarios, and interface wireframes. Initial validation was conducted via visual-focused A/B testing (layout, icons, information hierarchy) of two design variants.Key findings reveal:Non-technical users (administrative staff, students) prefer minimalist designs with LLM-powered interactive visual guides.Technical users (software engineers, data analysts) require flexibility for custom script integration within visual interfaces.LLMs show strong potential as mediators between natural language instructions and conceptual automation logic.This design is expected to lay the foundation for inclusive automation applications in Windows, with its core contribution being a user-centered design framework integrating visual elements, action recording, and generative LLM intelligence. Future work will focus on functional prototype implementation and quantitative usability testing*.*

***Keywords :***Interface Design, Windows Automation, UX Journey, Large Language Models (LLMs), Multi-level Users, Macro Recording*.*

**INTRODUCTION**

Windows automation tools are designed to streamline repetitive tasks, enhance productivity, and reduce human error by automating interactions with software applications. These tools can range from simple macro recorders to advanced scripting languages and robotic process automation (RPA) systems.

A wide range of tools and technologies have been developed to support task automation in the Windows environment. Under Macro Recorders and Scripting Tools, AutoIt is a free scripting language that simulates mouse movements, keystrokes, and other user actions, making it ideal for automating repetitive tasks and interacting with applications as a user would [1], [2]. AutoHotkey similarly allows for task automation through simulated keyboard and mouse actions, and is known for being user-friendly, even for those without advanced programming skills [3]. PowerShell, built on the .NET framework, is a command-line shell and scripting language designed for automating administrative tasks and integrating with Windows components [4]. In the area of Robotic Process Automation (RPA), tools described in various studies can automate complex workflows by mimicking human interaction with software applications. These tools are often platform-dependent and use rule-based logic to determine when tasks should be offloaded to edge or cloud processing [5], [6], [7]. For GUI Automation, tools like Korat leverage computer vision to automate GUI interactions across different platforms, particularly for repetitive mouse and keyboard operations [8], [9]. Assistgui serves as a benchmark for evaluating models that manipulate the mouse and keyboard on the Windows platform, showcasing the potential of multi-agent collaboration frameworks [10]. Lastly, in Web Automation, Ringer is a record-and-replay tool that allows non-coders to automate web interactions by generating scripts from user demonstrations and remains robust against changes in webpage implementations [11].

Automation tools have a wide range of practical applications across various domains. In productivity software, tools like AutoIt and AutoHotkey are frequently used to streamline workflows in applications such as Microsoft Office and Adobe Creative Cloud, enabling users to bind repetitive functions to keyboard and mouse macros for increased efficiency [12], [13]. In the field of laboratory automation, scripting languages help automate interactions with instrument control software, significantly reducing the need for manual input and minimizing human errors [3], [14]. For administrative tasks, PowerShell and other scripting tools are widely employed to automate routine processes within Windows environments, enhancing operational efficiency and alleviating the workload on IT professionals [4], [15].

Despite their benefits, automation tools still face several challenges and offer opportunities for future improvement. Integration and compatibility remain significant concerns, as many automation tools must work across diverse software applications and platforms. While tools like AutoHotkey and AutoIt provide broad compatibility with vendor software packages, seamless integration is still an area for enhancement [2], [3]. User accessibility is another critical issue—making these tools more approachable for non-programmers is essential. Efforts from tools like Ringer and AutoHotkey aim to lower the barrier to entry through user-friendly interfaces and flexible scripting features [3], [11]. Finally, performance and reliability are vital, particularly as software interfaces evolve. Tools such as Ringer and Assistgui are designed to be resilient to changes in interface structures, but continued research is required to further improve their robustness and consistency [10], [11].

In summary, Windows automation tools encompass a wide range of technologies designed to automate repetitive tasks, improve productivity, and reduce errors. While significant progress has been made, challenges related to integration, user accessibility, and reliability remain areas for future development.

**LITERATURE REVIEW**

This literature review synthesizes prior work that frames the present study on Windows automation interfaces. We organize the review around three interrelated themes that recur across the cited studies: (1) scripting-based automation tools (e.g., AutoIt, AutoHotkey, PowerShell), which provide low-level control but often require programming knowledge; (2) Robotic Process Automation (RPA) frameworks that enable higher-level workflow orchestration but raise issues of platform dependence and integration complexity; and (3) vision- and demonstration-based approaches (including tools and benchmarks such as Korat, AssistGUI, and demonstration-to-script systems like Ringer) that aim to improve robustness across changing interfaces and lower the barrier to entry for non-technical users. For each theme we summarize main findings, common methodologies, and persistent limitations particularly in terms of usability for multi-level users, resilience to UI changes, and the absence of integrated AI-assisted script generation thereby motivating the design choices and research questions addressed in this paper.

**Scripting-Based Automation Tools**

A variety of scripting-based tools have been developed to automate user interactions in Windows environments. Among the most widely adopted are AutoIt and AutoHotkey. AutoIt is a powerful scripting language capable of simulating mouse movements, keystrokes, and GUI interactions, making it suitable for repetitive task automation and application-level scripting [1], [2]. AutoHotkey offers similar capabilities but emphasizes ease of use for non-technical users, allowing simple automation through intuitive syntax and hotkey binding [3]. PowerShell, on the other hand, is a command-line shell and scripting language that leverages the .NET framework to automate administrative tasks and manage system configurations [4].

**Robotic Process Automation (RPA)**

Robotic Process Automation (RPA) introduces more sophisticated automation through rule-based logic and decision-making capabilities. Unlike traditional scripting tools, RPA systems can automate end-to-end business workflows and are frequently used in enterprise contexts. Chuan et al. [5] and Nakić & Boban [6] demonstrated RPA’s utility in automating tasks in educational and administrative domains, though platform dependence and integration complexity remain significant barriers [7].

**GUI Automation and Computer Vision**

In GUI automation, computer vision techniques have enabled new ways to interact with screen elements. Tools like Korat and AssistGUI employ visual recognition to simulate user actions across different software interfaces [8], [9], [10]. AssistGUI in particular introduces a multi-agent collaboration model that allows task-oriented GUI automation, demonstrating a promising direction for scalable desktop automation.

**Demonstration-Based and Web Automation**

Tools like Ringer bring a novel approach by allowing users to demonstrate tasks that are then automatically converted into scripts. This record-and-replay mechanism provides accessibility for non-programmers by eliminating manual coding [11]. Ringer also features robustness to changes in web structure, which is a key challenge in web automation.

**Practical Applications of Automation**

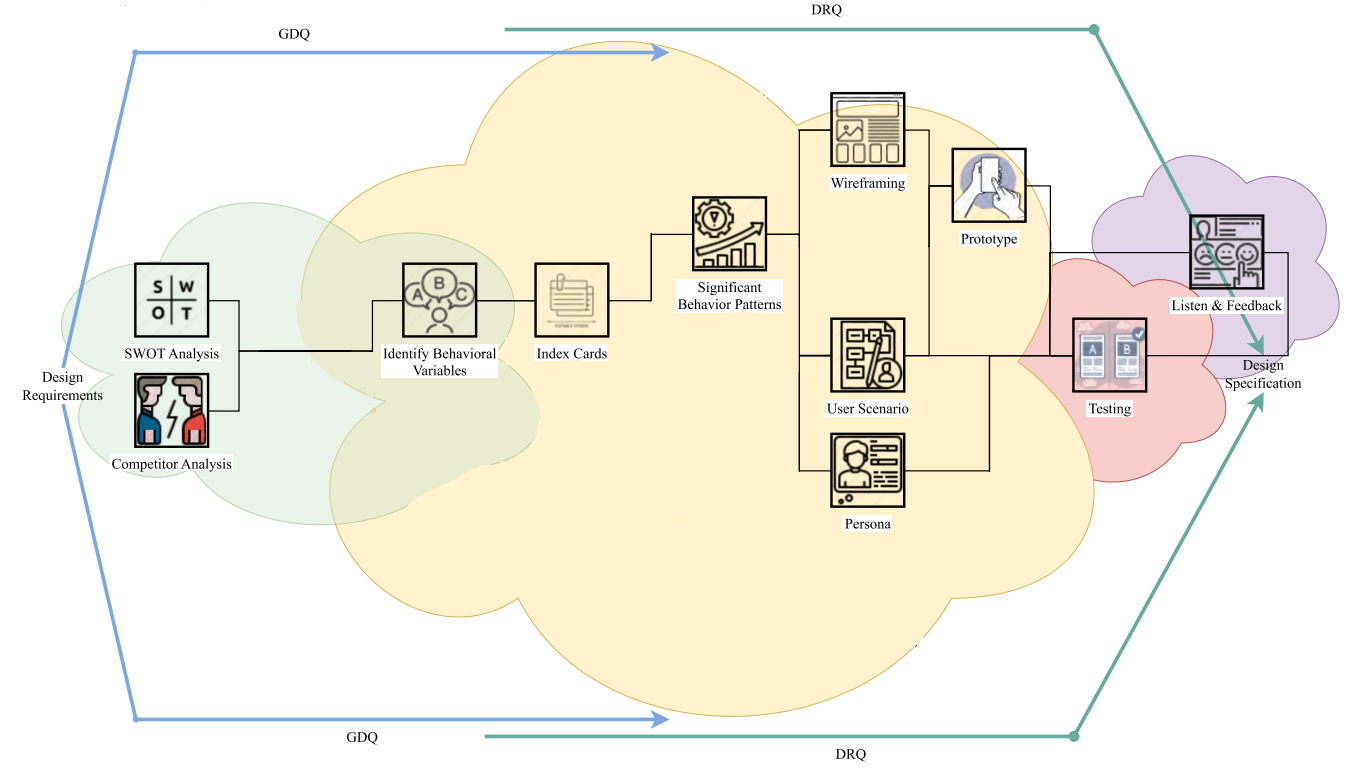
Automation tools are applied in a wide range of contexts. In productivity software, AutoIt and AutoHotkey streamline repetitive tasks in Microsoft Office and Adobe Creative Suite through macro bindings [12], [13]. In laboratory environments, scripting tools automate instrument control software, minimizing human input and reducing error [3], [14]. For administrative tasks, PowerShell remains the go-to choice for system automation in Windows environments [4], [15].

**Challenges and Opportunities**

Despite their success, automation tools face persistent challenges, particularly in platform compatibility, user accessibility, and reliability under changing UI conditions. Enhancing integration across diverse systems and simplifying scripting for non-technical users remain pressing goals. Tools like Ringer and AssistGUI exemplify ongoing efforts to build resilient and user-friendly automation systems [10], [11]. Therefore, integrating AI-assisted scripting as proposed in this study offers a promising pathway to make automation more intuitive and accessible, especially for end-users unfamiliar with programming.

**RESEARCH METHODS**

**From Figure 1**. This study adopts the UX Journey framework, which consists of four main stages: Discover, Explore, Test, and Listen [12], [15]. A mixed-methods approach was applied, integrating qualitative user research with quantitative validation, supported by Large Language Models (LLMs) for analysis and synthesis throughout the process



**Figure 1.**research flow

**Discover**

The Discover stage aimed to understand user needs, context, and the core design challenges. Activities included a SWOT analysis and competitor analysis supported by LLMs to identify strengths, weaknesses, opportunities, and threats in Windows automation tools (following market and usability evaluation practices in [5], [6], [7]). The outcome of this stage was the primary hypothesis: Users with minimal technical background can automate repetitive tasks effectively if supported by an intuitive interface and AI-assisted script generation.

**Explore**

This stage focused on generating and refining design concepts through co-design sessions with five multi-level participants (both technical and non-technical), aligning with participatory design principles in [12], [13]. Key outputs included user personas, scenarios, wireframes, and a sitemap developed using Figma. LLMs assisted in synthesizing interview data and constructing interaction scenarios.

**Test**

Testing was conducted through A/B testing (based on evaluation methods in [10], [11]) comparing two interface variants: (1) a minimalist icon-based UI and (2) a text-based UI with detailed descriptions. Performance metrics included task completion time, error rate, and qualitative user feedback.

**Listen**

The Listen stage focused on analyzing feedback and refining the design, in line with user-centered design principles [14], [15]. LLMs were employed to detect recurring needs and usability issues, which informed the next iteration of the interface.

**RESULTS AND DISCUSSION**

This section presents the outcomes of each UX Journey stage, connecting the methodological approach to the findings. The results are discussed in terms of how each stage informed design decisions and validated the initial hypothesis.

**Discover**

The SWOT analysis indicated that the proposed application’s main strengths were ease of use, integration with AI assistants, and a modular macro system. Weaknesses included the absence of built-in scripting documentation and potential performance constraints on older systems. Opportunities stem from the growing demand for no-code/low-code automation tools, while threats came from established competitors such as Jitbit and Pulover's Macro Creator [5], [6]. The competitor analysis confirmed the uniqueness of the integrated AI scripting assistant compared to existing solutions..

**Explore**

The co-design process resulted in three key user personas:

* Non-technical users (administrative staff, students) preferred minimalist designs with interactive, AI-powered visual guidance.
* Semi-technical users desired flexibility to modify simple scripts.
* Technical users (software engineers, data analysts) required full custom script integration within the visual interface.

Wireframes and the sitemap revealed that non-technical users navigated drag-and-drop workflows more intuitively, while technical users valued the script preview panel.

**Test**

The A/B testing revealed that the icon-based UI led to 20% faster task completion and 30% fewer errors compared to the text-based UI [10], [11]. Participants also highlighted the macro preview feature and contextual AI assistance as the most useful elements.

**Listen**

Feedback analysis identified two major improvement areas: (1) enhancing script flexibility for advanced users, and (2) adding a community-driven macro template library. These findings align with prior studies emphasizing the role of shared resources in improving automation adoption [12], [13]. These improvements will guide future prototype development.

**CONCLUSION**

This study demonstrates that integrating AI-assisted scripting within a user-friendly macro automation interface can significantly improve accessibility for novice users while still meeting the flexibility needs of technical users. By applying the four-stage UX Journey Discover, Explore, Test, and Listen the research systematically identified user needs, developed and refined interface concepts, validated design choices through comparative testing, and incorporated user feedback for iterative improvement.  
The findings confirm that a minimalist icon-based interface, combined with contextual AI guidance and macro preview, enhances task efficiency and reduces user error rates. Future development will focus on expanding scripting flexibility and establishing a shared macro template library to foster a community-driven ecosystem for Windows automation.

**ACKNOWLEDGMENTS**

The author would like to express sincere gratitude to all participants for their valuable time and insights during the design and testing phases. Special thanks are extended to the academic supervisor for constructive feedback throughout the research process, the UX mentors and technical advisors for their guidance, and the LLM assistant whose capabilities accelerated several research activities. Appreciation is also given to the institution for providing access to the necessary tools and resources that enabled this study.

**REFERENCES**

[1] S. E. White III, “Automating manual procedures with Autolt,” presented at the 35th International Conference Computer Measurement Group, 2009. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84872157727&partnerID=40&md5=2e68cac963cfbf0d794c6169139e80bf

[2] J. Brand and J. Balvanz, “Automation is a breeze with AutoIt,” presented at the Proceedings ACM SIGUCCS User Services Conference, 2005, pp. 12–15. doi: 10.1145/1099435.1099439.

[3] S. Tentarelli, R. Romero, and M. L. Lamb, “Script-based automation of analytical instrument software tasks,” *SLAS Technol.*, vol. 27, no. 3, pp. 209–213, 2022, doi: 10.1016/j.slast.2021.10.019.

[4] H. Horban, I. Kandyba, A. Boiko, K. Kirey, and V. Chorna, “Windows powershell research from the point in terms of operational data analysis subsystem constructing,” presented at the CEUR Workshop Proceedings, 2019, pp. 69–84. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077179647&partnerID=40&md5=506211d808886535e2d9385978366fd9

[5] W. C. Chuan, S. Manickam, Y. W. Seng, and K. Y. K. Wen, “Enhance Robotic Process Automation Using a Rule-Based Approach,” presented at the Lecture Notes in Electrical Engineering, 2022, pp. 125–130. doi: 10.1007/978-981-16-8129-5\_20.

[6] J. Nakić and D. Boban, “Enhancing Informal Education with Robotic Process Automation: A Method for Automating Course Search on E-Learning Platforms,” *J. Commun. Softw. Syst.*, vol. 20, no. 2, pp. 215–225, 2024, doi: 10.24138/jcomss-2023-0178.

[7] S. Agostinelli, A. Marrella, and M. Mecella, “Exploring the Challenge of Automated Segmentation in Robotic Process Automation,” presented at the Lecture Notes in Business Information Processing, 2021, pp. 38–54. doi: 10.1007/978-3-030-75018-3\_3.

[8] M. Narayana, N. Raghu Ram Reddy, and N. Hyndavi Reddy, “High speed script execution for GUI Automation using Computer Vision,” *Int. J. Electr. Comput. Eng.*, vol. 9, no. 1, pp. 231–236, 2019, doi: 10.11591/ijece.v9i1.pp231-236.

[9] Y.-P. Cheng, C.-W. Li, and Y.-C. Chen, “Apply computer vision in GUI automation for industrial applications,” *Math. Biosci. Eng.*, vol. 16, no. 6, pp. 7526–7545, 2019, doi: 10.3934/mbe.2019378.

[10] D. Gao *et al.*, “AssistGUI: Task-Oriented PC Graphical User Interface Automation,” presented at the Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2024, pp. 13289–13298. doi: 10.1109/CVPR52733.2024.01262.

[11] S. Barman, S. Chasins, R. Bodik, and S. Gulwani, “Ringer: Web automation by demonstration,” presented at the Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA, 2016, pp. 748–764. doi: 10.1145/2983990.2984020.

[12] A. Jalil, R. Kolandaisamy, K. Subaramaniam, I. Kolandaisamy, and J. Q. G. Khang, “Designing a mobile application to improve user’s productivity on computer-based productivity software,” *J. Adv. Res. Dyn. Control Syst.*, vol. 12, no. 3 Special Issue, pp. 226–236, 2020, doi: 10.5373/JARDCS/V12SP3/20201257.

[13] S. N. Lee *et al.*, “Quick Guide: Programming a Gaming Keyboard for PACS to Optimize Radiology Workflow,” *Curr. Probl. Diagn. Radiol.*, vol. 53, no. 1, pp. 1–16, 2024, doi: 10.1067/j.cpradiol.2023.09.002.

[14] M. C. Carvalho, “Integration of Analytical Instruments with Computer Scripting,” *J. Lab. Autom.*, vol. 18, no. 4, pp. 328–333, 2013, doi: 10.1177/2211068213476288.

[15] A. Ospanova, B. Tuleuov, G. Karzhauova, and L. Kussepova, “Advanced Administration of Windows Based on Open Source Utilities,” presented at the Lecture Notes in Engineering and Computer Science, 2021, pp. 118–121. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85125635265&partnerID=40&md5=c63a14a3aa02b5d3acce0079872df01c