Lindenwood University
Digital Commons@Lindenwood University

Faculty Scholarship | Research and Scholarship

8-2024

Beyond Automation: AI as a Catalyst for New Job Creation in Software Development

Jill Willard

James Hutson

Follow this and additional works at: https://digitalcommons.lindenwood.edu/faculty-research-papers

 Part of the Artificial Intelligence and Robotics Commons

International Journal Of Engineering Research And Development
e- ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 20, Issue 8 (August, 2024), PP. 419-424

Beyond Automation: AI as a Catalyst for New Job Creation in Software Development

Jill Willard, *CTO XplorPay, Caledonia, IL, USA*
James Hutson, Lindenwood University, USA https://orcid.org/0000-0002-0578-6052
Corresponding author: Jill Willard

Abstract
As artificial intelligence (AI) continues to evolve, its impact on software development and programming is profound, drawing parallels to the shift from assembler to object-oriented programming. This article explores how AI is reshaping the landscape of software jobs, creating new opportunities rather than diminishing them. By simplifying complex tasks and lowering barriers to coding, AI is expanding the technology "pie," introducing new use cases, and enhancing efficiency. The transition from monolithic services to microservices has reduced risks and accelerated deployment processes, and AI is poised to further this evolution by managing the complexities of service interactions through advanced orchestration layers. Despite fears of job displacement, AI is likely to generate new roles in overseeing and integrating these systems, much like previous technological shifts. The article also underscores the importance of continuous education and skill retooling in the AI-driven future, advocating for more accessible and affordable higher education to equip the workforce with durable skills. As AI continues to integrate into the software industry, it will require human oversight to navigate and manage its complexities, ensuring that the future job market remains robust and dynamic. This article ultimately positions AI not as a job-reducing force, but as a catalyst for expanding opportunities in the software industry, emphasizing the necessity of adapting to and embracing this technological advancement.

Keywords: *AI in software development, microservices, job market expansion, continuous deployment, skill retooling*

Date of Submission: 12-08-2024 Date of Acceptance: 27-08-2024

I. Introduction

Generative AI has significantly reshaped industries across the globe, influencing everything from creative work to business operations. Generative AI (GAI), such as OpenAI's GPT models, has the potential to transform tasks that were previously considered immune to automation, by not just enhancing efficiency but also by augmenting the creative process in fields like marketing, content creation, and design (Zohuri, 2023). The capacityof technology to generate content autonomously is driving innovation while also introducing new business models that emphasize scalability and personalization. This transformation is especially impactful in knowledge-based sectors, where the role of GAI in automating routine tasks can free up human workers to focus on more strategic, creative, and high-value activities (Brynjolfsson et al., 2023).

However, the widespread adoption of GAI has sparked debates about its implications for employment. While some fear it could lead to job displacement, recent studies suggest that the net effect may be job augmentation rather than automation, particularly in high-income regions where clerical and routine cognitive roles are more prevalent (Gmyrek et al., 2023). This augmentation primarily benefits less experienced or lower-skilled workers by disseminating expert knowledge more effectively and streamlining processes. Nonetheless, it is clear that AI is accelerating shifts in labor markets, necessitating proactive reskilling and upskilling efforts to ensure that the workforce adapts to these technological changes (Olaniyi et al., 2024).

Since the 1980s, software development began undergoing a significant evolution, particularly transitioning from mainframe systems to object-oriented programming (OOP) (Nagineni, 2021). In the mainframe era, development was highly centralized, with monolithic applications being managed by a few highly specialized professionals (Megargel, Shankararaman, & Walker, 2020). However, as the industry shifted towards object-oriented paradigms, the landscape of programming changed dramatically. Object-oriented programming introduced modularity, encapsulation, and reusability, which simplified the development process and allowed for more scalable and maintainable code (Saide, 2024). This transition not only enabled more complex applications to be built but also opened up software development to a broader range of professionals, reducing the steep learning curve traditionally associated with programming (Gutiérrez, Guerrero, & López-Ospina, 2022; Li et al., 2008).

The shift towards object-oriented programming brought with it a new set of challenges and opportunities. As object-oriented design became the standard, it required developers to acquire a different skill set focused on understanding class hierarchies, inheritance, and polymorphism. These concepts are fundamental to creating flexible and maintainable software, but they also introduced complexities, particularly when dealing with evolving software requirements (Jablonický& Lang, 2023). For instance, evolving class hierarchies and managing dependencies between objects became critical aspects of maintaining large-scale systems (Kasauli et al., 2021). Consequently, software engineers increasingly relied on methodologies and tools that supported the iterative evolution of object-oriented systems, such as refactoring and design pattern applications (Rajlich, 1997). This evolution underscores how skills in software development have continuously adapted, allowing programmers to address growing system complexities while maintaining efficiency and scalability.

As AI continues to integrate into the software industry, these recent developments indicate that it will reshape job roles and daily activities rather than completely replace software jobs. The progression of AI in the workplace reveals that automation is more likely to impact specific tasks within roles, leading to augmentation rather than outright replacement. For instance, AI tools are designed to handle repetitive and routine tasks, freeing up human workers to focus on creative, strategic, and complex problem-solving activities (Santhosh et al., 2023). This shift mirrors past technological changes, where the introduction of new tools and methodologies reduced the burden of routine tasks and allowed professionals to engage in higher-value work.

As with previous technological advancements, such as the transition from mainframe systems to object-oriented programming, AI is expected to create new roles and opportunities rather than diminish them. These roles will likely involve overseeing AI-driven processes, integrating systems, and ensuring that AI applications are used effectively and ethically. While concerns remain about potential job displacement, evidence suggests that AI will more likely expand the job market by introducing new areas of expertise, particularly in managing AI systems and orchestrating complex service interactions (Tolan et al., 2021). As such, the software industry will see a shift in daily activities, with a stronger focus on continuous learning and adapting to new AI-driven tools and practices.

Software Development Evolving

The history of software development is a narrative of rapid technological progress, characterized by distinct eras that shaped the growth of the industry. Beginning in the 1940s and 1950s with the development of early computers, the field has transitioned from rudimentary machine language coding to the more sophisticated programming paradigms we see today. This progression involved significant shifts, such as moving from mainframe systems with batch processing to more modular and flexible development methodologies like object-oriented programming. Each period introduced advancements that reduced the complexity of coding and broadened access to software development, eventually leading to today's highly interconnected and automated systems (Jadhav, Kaur & Akter, 2022).

The first generation of computers developed in the 1940s, including systems like ENIAC, were primarily designed for scientific and military applications (Haigh & Ceruzzi, 2021). Programming was done using machine language and assembly, with instructions inputted through punch cards (Arawjo, 2020). The Von Neumann architecture, introduced during this period, revolutionized computing by introducing the concept of stored programs, allowing instructions to be kept in memory for sequential execution (Collen & Kulikowski, 2015). These early computers were massive, costly, and limited in functionality but laid the groundwork for future advancements in both hardware and software.

As computing technology advanced, the mainframe era began, marked by the dominance of large-scale computers used primarily by governments and large corporations. During this time, programming languages like COBOL and Fortran were developed to handle business and scientific applications, respectively (Bessen, 2022). The structured programming principles introduced in the 1960s helped to manage the increasing complexity of software systems, providing a foundation for more maintainable and efficient code (Farley, 2021). Mainframes operated on batch processing, where tasks were queued and executed sequentially, which limited interactivity but supported large-scale data processing needs (Campbell-Kelly & Garcia-Swartz, 2015). This era also saw the beginnings of standardization in software development practices, setting the stage for more flexible computing systems. These early periods highlight the foundational shifts in software development, from limited, specialized systems to broader, more accessible programming practices that have continuously evolved to meet new technological demands (Kasauli et al., 2021).

The 1970s and 1980s marked a significant shift in software development with the advent of personal computing, driven largely by the development of microprocessors (Khan et al., 2021). As computing power became more affordable and accessible, personal computers (PCs) began to enter homes and offices. This era saw the popularization of operating systems like MS-DOS and the widespread use of programming languages like BASIC, which made computing more approachable for hobbyists and professionals alike (Bright et al., 2020). The introduction of graphical user interfaces (GUIs) with products like Apple's Macintosh and Microsoft Windows revolutionized software usability, making computers intuitive for non-technical users and expanding

the user base significantly (Ceruzzi, 1998). These developments paved the way for the personal computing boom, transforming the software industry by shifting focus from mainframes to more user-centric applications (Barlaskar, 2020).

The 1980s also saw the emergence of object-oriented programming (OOP), a paradigm that introduced concepts such as encapsulation, inheritance, and polymorphism (Koti et al., 2024). These concepts allowed developers to create more modular, maintainable, and scalable software systems. Languages like C++ and later Java became dominant, allowing complex applications to be built with greater flexibility and efficiency (Ogala& Ojie, 2020). OOP fundamentally changed software development by shifting the focus from procedural programming to a more object-based approach, where software components could be reused and managed more effectively (Dony et al., 1992). The rise of the client-server model during this time further enabled distributed applications, which allowed businesses to run enterprise-level software across interconnected systems, driving further adoption of OOP methodologies (Sallow et al., 2020). These periods illustrate how the convergence of accessible personal computing and innovative programming paradigms like OOP set the stage for the rapid expansion of software development, leading to the diverse and interconnected systems we rely on today.

The 1990s and early 2000s saw a transformative period in software development with the rapid growth of the internet and the rise of open-source software. The widespread adoption of web technologies like HTML, JavaScript, and PHP enabled the development of dynamic and interactive websites, leading to the proliferation of web-based applications (Lendarduzzi et al., 2020). Software such as web browsers, email clients, and early content management systems became essential tools as the internet became more ingrained in everyday life. This era also marked the emergence of collaborative, community-driven development models in software, most notably through the open-source movement (Tabarés, 2021). Projects like Linux and the Apache HTTP server were pivotal, showcasing how decentralized development could produce reliable and scalable software. These open-source initiatives not only fueled innovation but also challenged traditional software business models by making software freely available and modifiable (Bretthauer, 2001).

During the 2000s, software development methodologies underwent significant shifts with the introduction of Agile practices (Argen et al., 2022). Moving away from the rigid, sequential waterfall model, Agile methodologies emphasized iterative development, continuous feedback, and close collaboration with customers. Agile allowed teams to quickly adapt to changing requirements and deliver software in small, manageable increments, significantly improving productivity and customer satisfaction (Ogundipe et al., 2024). This period also saw the emergence of DevOps, a cultural shift that integrated development and operations to streamline the deployment process. DevOps practices focused on automating the entire software delivery pipeline, enabling continuous integration and continuous delivery (CI/CD) (Mishra & Otaiwi, 2020). By breaking down silos between teams and promoting automation, organizations were able to deploy updates more frequently and with greater reliability (Mockus et al., 2002).These advancements highlight how the combination of Agile, DevOps, and open-source development has fundamentally reshaped software engineering, allowing for faster iteration, improved collaboration, and more resilient systems.

Future of Software in the Age of AI

The integration of AI into software development is revolutionizing how code is written, tested, and deployed. AI-powered tools such as GitHub Copilot have become increasingly popular, offering developers automated code suggestions and autocompletion that can significantly enhance productivity. These tools leverage large language models (LLMs) trained on extensive code repositories to generate relevant code snippets based on natural language inputs. Research indicates that developers primarily use these tools to reduce keystrokes, complete tasks faster, and recall syntax, making them valuable for both novice and experienced programmers. However, challenges remain, including limitations in the functional accuracy of generated code and the cognitive overhead required to validate AI-generated suggestions (Liang et al., 2023).

Continuous deployment practices have also been enhanced by AI-driven automation tools. Modern software engineering emphasizes rapid, small, and incremental changes, facilitated by CI/CD pipelines and orchestration tools. AI supports these processes by automating testing, deployment, and monitoring, thereby reducing the need for manual intervention and enabling more frequent releases. This automation reduces the risk associated with updates by ensuring that only validated and tested code is deployed. For instance, automated deployment pipelines integrated with AI can handle everything from code commits to production deployment, allowing for seamless updates with minimal downtime. As a result, companies can achieve greater agility and faster time-to-market without sacrificing reliability (Sailer & Petrič, 2019).

Despite these advancements, the implementation of AI in software development is not without its complexities. Developers have raised concerns about issues such as compatibility and integration challenges when using AI tools like GitHub Copilot. While these tools excel in generating code, there are still significant hurdles to overcome in terms of usability and integration within existing workflows. AI assistants are expected to evolve, focusing on improving the quality of suggestions and reducing the cognitive load on developers.

Continued research and development will likely refine these tools, making them more reliable and effective, thereby solidifying AI's role in the future of software engineering (Zhou et al., 2023).

Furthermore, GAI tools are rapidly expanding their capabilities, enabling more professionals without traditional programming backgrounds to perform complex software development tasks. Tools like GitHub Copilot and ChatGPT allow users to generate functional code from natural language prompts, which significantly lowers the barrier to entry for those who are not formally trained in coding. By automating code generation, bug detection, and even deployment processes, GAI tools make it easier for professionals in fields like design, marketing, and data analysis to integrate software development into their workflows. For example, in web development, GAI tools are already being used to create and modify website components without requiring deep knowledge of HTML, CSS, or JavaScript. This democratization of software development means that more industries can incorporate custom solutions tailored to their specific needs, driven by professionals who understand their domain but are not necessarily coders (Bull & Kharrufa, 2023).

The implications of this trajectory extend beyond merely automating routine coding tasks. GAI systems are increasingly being used in creative and strategic roles, offering non-programmers the ability to prototype applications, automate data analysis, and even develop AI models. For instance, in innovation management and digital prototyping, GAI tools are being leveraged to rapidly iterate designs and generate diverse solutions, empowering professionals without coding expertise to directly engage in technical processes. This historical trend suggests a future where software development becomes a collaborative, cross-disciplinary activity, supported by AI tools that handle the technical complexity**(Table 1)**. Such tools not only enhance productivity but also reduce the need for specialized coding knowledge, allowing more professionals to focus on high-level problem-solving and innovation (Ebert et al., 2023).

Table 1. Evolution of Software Development

Era	Timeline	Processes/Technologies	Skills/Barriers to Entry	Significance
Early Days	1940s-1950s	Machine language, Assembly, Punch cards, Von Neumann architecture	Highly specialized skills, limited access, steep learning curve	Foundation of digital computing and stored-program concept (Collen & Kulikowski, 2015).
Mainframe Era	1950s-1970s	Batch processing, COBOL, Fortran, Structured programming	Centralized, large-scale systems; specialized knowledge required	Standardized processes, large business and government use (Bessen, 2022)
Rise of Personal Computing	1970s-1980s	Microprocessors, GUIs, Operating systems (MS-DOS, Windows)	Lower barrier to entry with BASIC and accessible hardware	Widespread use of personal computers and user-centric design (Khan et al., 2021)
Object-Oriented Programming	1980s-1990s	C++, Java, Encapsulation, Inheritance, Polymorphism	Understanding OOP concepts, class hierarchies, modular systems	Modular and maintainable software, distributed systems (Dony et al., 1992)
Internet and Open Source Era	1990s-2000s	Web technologies (HTML, JavaScript, PHP), Open Source (Linux, Apache)	Collaborative development, community-driven contributions	Decentralized software innovation, web application boom (Lenarduzzi et al., 2020)
Agile and DevOps	2000s-2010s	Agile methodologies, DevOps, CI/CD pipelines	Cross-functional collaboration, iterative development, automation	Faster iteration, improved quality and reliability (Mishra & Otaiwi, 2020)
Cloud Computing and Microservices	2010s-Present	Cloud infrastructure (AWS, Azure), Microservices architecture	Cloud orchestration, independent services, scalability	Enhanced flexibility, resilient architectures (Sailer & Petrič, 2019)
AI and Automation in Software Development	Present	AI-assisted development (GitHub Copilot, automated testing), Continuous Deployment	AI integration, minimal coding knowledge, strategic oversight	Democratization of software development, cross-disciplinary collaboration (Bull &Kharrufa, 2023)

II. Conclusion

The rapid advancements in AI have had significant implications for software development, transforming traditional coding practices and introducing new paradigms for automation and deployment. As AI-driven tools like GitHub Copilot and ChatGPT become more sophisticated, they allow for greater efficiency in coding tasks by offering automated code suggestions and accelerating development cycles. This integration of AI is not only enhancing productivity but also reshaping the skill sets required in software engineering. The shift from monolithic architectures to microservices and continuous deployment models has been further streamlined by AI technologies that manage complex orchestration tasks, reducing both the time and risk associated with software releases (Bull &Kharrufa, 2023).

The significance of these developments lies in how they redefine the software industry's labor landscape. While concerns about job displacement persist, the evidence suggests that AI will likely augment rather than replace software roles. New opportunities are emerging for professionals with interdisciplinary skills who can harness AI tools without extensive programming knowledge, thereby expanding the talent pool in technology-driven sectors. Moreover, the democratization of software development through AI is lowering entry barriers, allowing professionals from various backgrounds to contribute to coding, design, and system management without requiring deep technical expertise (Ebert et al., 2023).

Looking ahead, the trajectory of software jobs in the AI era suggests a dynamic shift toward roles that emphasize oversight, integration, and strategic use of AI systems rather than manual coding. The complexities of AI-driven automation may reduce the demand for traditional software engineers while increasing the need for specialists in AI ethics, data management, and system integration. Nonetheless, job displacement will vary across sectors, with routine and repetitive tasks being the most vulnerable. The future of work in software development is thus characterized by a symbiotic relationship between human creativity and machine efficiency, where continuous learning and adaptability remain crucial for professionals to thrive in an AI-enhanced environment (Karangutkar, 2023).

Data Availability

Data available upon request.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Funding Statement

NA

Authors' Contributions

Conceptualization, J. Willard; Methodology, J. Willard; Validation, J. Wilard; Investigation, J. Willard – Original Draft Preparation, J. Hutson; Writing – Review & Editing, J. Hutson.; Visualization, J. Hutson.

References

[1]. Arawjo, I. (2020, April). To write code: The cultural fabrication of programming notation and practice. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (pp. 1-15).
[2]. Ågren, P., Knoph, E., & Berntsson Svensson, R. (2022). Agile software development one year into the COVID-19 pandemic. Empirical Software Engineering, 27(6), 121.
[3]. Barlaskar, E. (2020). User-centric cloud application management (Doctoral dissertation, Queen's University Belfast).
[4]. Bessen, J. (2022). The new goliaths: How corporations use software to dominate industries, kill innovation, and undermine regulation. Yale University Press.
[5]. Bretthauer, D. (2001). Open Source Software: A History. Information Technology and Libraries, 21, 3-10.
[6]. Bright, W., Alexandrescu, A., & Parker, M. (2020). Origins of the D programming language. Proceedings of the ACM on Programming Languages, 4(HOPL), 1-38.
[7]. Brynjolfsson, E., Li, D., & Raymond, L. R. (2023). Generative AI at work (No. w31161). National Bureau of Economic Research.
[8]. Bull, C., &Kharrufa, A. (2023). Generative AI Assistants in Software Development Education. ArXiv, abs/2303.13936. https://doi.org/10.1109/MS.2023.3300574
[9]. Campbell-Kelly, M., & Garcia-Swartz, D. D. (2015). From mainframes to smartphones: a history of the international computer industry. Harvard University Press.
[10]. Ceruzzi, P. E. (2003). A history of modern computing. MIT press.
[11]. Collen, M. F., & Kulikowski, C. A. (2015). The development of digital computers. The History of Medical Informatics in the United States, 3-73.
[12]. Dony, C., Purchase, J., & Winder, R. (1992). Exception handling in object-oriented systems. ACM SIGPLAN OOPS Messenger, 3(2), 17-30.
[13]. Ebert, C., Louridas, P., & Ebert, C. (2023). Generative AI for Software Practitioners. IEEE Software, 40, 30-38. https://doi.org/10.1109/MS.2023.3265877
[14]. Farley, D. (2021). Modern Software Engineering: Doing What Works to Build Better Software Faster. Addison-Wesley Professional.
[15]. Gmyrek, P., Berg, J., & Bescond, D. (2023). Generative AI and jobs : a global analysis of potential effects on job quantity and quality. ILO working papers.https://doi.org/10.54394/fhem8239
[16]. Gutiérrez, L. E., Guerrero, C. A., & López-Ospina, H. A. (2022). Ranking of problems and solutions in the teaching and learning of object-oriented programming. Education and Information Technologies, 27(5), 7205-7239.
[17]. Haigh, T., & Ceruzzi, P. E. (2021). A new history of modern computing. MIT Press.
[18]. Jablonický, K., & Lang, J. (2023). Code Based Selected Object-Oriented Mechanisms Identification. Proceedings http://ceur-ws. org ISSN, 1613, 0073
[19]. Jadhav, A., Kaur, M., & Akter, F. (2022). Evolution of software development effort and cost estimation techniques: five decades study using automated text mining approach. Mathematical Problems in Engineering, 2022(1), 5782587.
[20]. Kasauli, R., Knauss, E., Horkoff, J., Liebel, G., & de Oliveira Neto, F. G. (2021). Requirements engineering challenges and practices in large-scale agile system development. Journal of Systems and Software, 172, 110851.
[21]. Khan, F. H., Pasha, M. A., & Masud, S. (2021). Advancements in microprocessor architecture for ubiquitous AI—An overview on history, evolution, and upcoming challenges in AI implementation. Micromachines, 12(6), 665.
[22]. Karangutkar, A. (2023). The Impact of Artificial Intelligence on Job Displacement and the Future of Work. International Journal of Advanced Research in Science, Communication and Technology. https://doi.org/10.48175/ijarsct-12096
[23]. Koti, A., Koti, S. L., Khare, A., & Khare, P. (2024). 1335 Beyond the Paradigm: Unraveling the Limitations of Object-Oriented Programming. Multifaceted approaches for Data Acquisition, Processing & Communication, 95.

[24]. Lenarduzzi, V., Taibi, D., Tosi, D., Lavazza, L., & Morasca, S. (2020, August). Open source software evaluation, selection, and adoption: a systematic literature review. In 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 437-444). IEEE.

[25]. Li, H., Huang, B., & Lu, J. (2008, June). Dynamical evolution analysis of the object-oriented software systems. In 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence) (pp. 3030-3035). IEEE.

[26]. Liang, J., Yang, C., & Myers, B. (2023). Understanding the Usability of AI Programming Assistants. ArXiv, abs/2303.17125.https://doi.org/10.48550/arXiv.2303.17125

[27]. Megargel, A., Shankararaman, V., & Walker, D. K. (2020). Migrating from monoliths to cloud-based microservices: A banking industry example. Software engineering in the era of cloud computing, 85-108.

[28]. Mishra, A., &Otaiwi, Z. (2020). DevOps and software quality: A systematic mapping. Computer Science Review, 38, 100308.

[29]. Mockus, A., Fielding, R. T., &Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology (TOSEM), 11(3), 309-346.

[30]. Nagineni, R. B. (2021). A Research on Object Oriented Programming and Its Concepts. International Journal, 10(2).

[31]. Ogala, J. O., & Ojie, D. V. (2020). Comparative analysis of c, c++, c# and java programming languages. GSJ, 8(5), 1899-1913.

[32]. Ogundipe, D. O., Odejide, O. A., &Edunjobi, T. E. (2024). Agile methodologies in digital banking: Theoretical underpinnings and implications for customer satisfaction. Open Access Research Journal of Science and Technology, 10(2), 021-030.

[33]. Olaniyi, O. O., Ezeugwa, F. A., Okatta, C., Arigbabu, A. S., &Joeaneke, P. (2024). Dynamics of the digital workforce: Assessing the interplay and impact of AI, automation, and employment policies. Automation, and Employment Policies (April 24, 2024).

[34]. Rajlich, V. (1997). MSE: A methodology for software evolution. Journal of Software Maintenance: Research and Practice, 9(2), 103-124.

[35]. Saide, M. (2024). Understanding Object-Oriented Development: Concepts, Benefits, and Inheritance in Modern Software Engineering. Benefits, and Inheritance in Modern Software Engineering (July 01, 2024).

[36]. Sailer, A., & Petrić, M. (2019). Automation and Testing for Simplified Software Deployment. EPJ Web of Conferences.https://doi.org/10.1051/EPJCONF/201921405019

[37]. Sallow, A. B., Dino, H. I., Ageed, Z. S., Mahmood, M. R., & Abdulrazaq, M. B. (2020). Client/Server remote control administration system: design and implementation. Int. J. Multidiscip. Res. Publ, 3(2), 7.

[38]. Santhosh, A., Unnikrishnan, r., Shibu, S., Meenakshi, K., & Joseph, G. (2023). AI impact on job autonation. International Journal of Engineering Technology and Management Sciences.https://doi.org/10.46647/ijetms.2023.v07i04.05

[39]. Tabarés, R. (2021). HTML5 and the evolution of HTML; tracing the origins of digital platforms. Technology in Society, 65, 101529.

[40]. Tolan, S., Pesole, A., Martínez-Plumed, F., Fernández-Macías, E., Hernández-Orallo, J., & Gómez, E. (2021). Measuring the occupational impact of AI: tasks, cognitive abilities and AI benchmarks. Journal of Artificial Intelligence Research, 71, 191-236.

[41]. Zhou, X., Liang, P., Zhang, B., Li, Z., Ahmad, A., Shahin, M., & Waseem, M. (2023). On the concerns of developers when using GitHub Copilot. arXiv preprint arXiv:2311.01020.

[42]. Zohuri, B. (2023). Charting the future. The synergy of generative AI, quantum computing, and the transformative impact on economy. Current Trends in Engineering Science, 3(7), 1-4.