Aiden Allen
CEN 4020
Te-yen Wu

Automation In Software Engineering

Machine learning has become increasingly popular within the sphere of software development in recent years. Artificial intelligence is currently being utilized within software development to automate software testing, assist programmers, and analyze a program's usability. This paradigm shift has led to more efficient planning and programming, granting expedited updates and development. Automation within software engineering may offer a resolution to increased workloads, as a result of rising digitalization. Consequently, many debates have sparked over the potential risks and challenges posed by artificial intelligence, particularly its impacts on employment. Within the discourse, we will examine the implications of automation within software development, encompassing the advantages, as well as the complex problems posed by employment-related risks.

Software engineering is a fast paced environment where change needs to occur rapidly. Automation has become crucial to the software development process, an example being aspect based sentiment analysis (ASBA). ASBA allows for extraction of user reviews by identifying different aspects of an app that are being reviewed and assigning a sentiment (Nissim 2017). ASBA can be used to boost the user experience as well as allow staff to recognize essential needed features. Automation may be advancing to other sectors of software engineering, such as UML documentation. With the birth of UML 2 comes the possibility of automating the UML process. UML automation promises to expedite the analysis phase of software engineering, as well as increase the usability of UML. The automation of UML decreased the risk of bugs, due to consistency in the UML standards as well as eliminating some potential human errors. This will lead to an increasingly reliable system, additionally many UML automation softwares generate documentation for the user. One example being, Sparx System , allowing for specified control over documentation generation. Sparx system additionally offers code generation (Sparx System). Code generation is quickly developing within the sphere of software engineering. Code generation assists programmers in solving complex tasks, as well as solving bugs that may be present within the source code. The most popular code generation tool is Github Copilot. A study by Arghavan Dakhel found that copilot produced more consistent code than students did. Students tended to utilize functions such as “reverse” and “pop” while copilot was consistent in its utilization of “append”. Github copilot performed similarly to students on given problems as well (Dakhel 2023). A similar study concluded that, Copilot generated correct lines of code 94% of the time. Similarly Chat GPT generated lines of code with 98% accuracy (Hansson 2023). Code generation may allow for more consistent code, as well as increased compatibility. ChatGPT allows for the conversion of code from one language to another, which may be instrumental in processes such as refactoring a code base. Following the development stage, Software testing is crucial to the development cycle because it ensures functionality of the software. Software engineers spend 35-50 percent of their time debugging (Britton 2013). Testing consumes 40 to 70 percent of the development’s life-span (Sharma 2014). Testing ensures the product meets the stakeholders requirements as well as quality standards. Automation within product testing is typically done following manual tests in order to ensure the validity of the program. Product testing automation completes tests quicker and more effectively than a human counterpart, while saving money and resources (Mahajan 2016). Automated tests

allow for reliable and reusable scripts, this enables more detailed test logs, leading to better product performance.

Despite its numerous advantages, Automation within Software development has drawbacks. For Example, while aspect-based sentiment analysis (ABSA) offers valuable insights into the nuances of opinions and sentiments expressed in text, it is fairly inaccurate in its predictions. As stated in, “The analysis found that about 40% of the misclassified samples is because without considering aspect information” (Jiang 2011). Additionally researchers found that over 82% of software testing could be automated using their technique. Their technique furthermore promises to reduce the cost of test automation (Thummalapenta 2012). UML automation comes with many technical challenges. In critical applications missing requirements or functions, that may be overlooked by automation, are essential to software development. In order to successfully implement UML automation on a large scale models need to ensure the requirements are specific enough to meet the developers needs, and additionally ensure that standard UML semantics are upheld (Ritala 2007). Github copilot faces a similar issue. While the lines of code are often correct they fail to correctly generate algorithms. A study assessed that Copilot only generated 64% of algorithms correctly with no quality rule violations (Hansson 2023). ChatGPT and Copilot are expected to become impactful tools used within software engineering. They are used throughout the world to assist programmers, however, without proper knowledge they leave room for code vulnerability as a result of their inaccuracies. Chat GPT and Copilot do provide useful tools for debugging. Researchers found that chatGPT’s ability to analyze code, find bugs, as well as fix bugs makes it an indispensable tool to use while debugging. ChatGPT provides a cost effective, accurate, as well as fast way to solve bugs that software engineers may struggle to find (Surameery 2022). This can allow for increased productivity coupled with early identification of bugs within software. Code generation is a promising step in ensuring continuous integration within software. Automation promises to ensure resource optimization, as well as provide a cost effective efficient solution to software engineering.

Although advances in automation within software engineering are promising they should not replace software engineers. For example, within product testing manual tests must first be in place due to automation tests having a higher impact on a product's execution. Automation should only be done on software once, manual tests pass, tested twice at minimum, and the project is maintained within a stable environment (Mahajan 2016). This calls into question the feasibility of automation within software engineering. Automation is difficult to implement within the software engineering process. The initial overhead of implementing automation is significant, requiring a significant investment in time as well as resources. Within small projects automation may not be feasible due to the high overhead, as well as the complexity within troubleshooting. Implementation additionally requires a steep learning curve for established developers. In turn, Under experienced developers may develop a false sense of security as a result of code generation. Automation requires maintenance, which may pose many issues if neglected. Neglection of maintenance may lead to obsolete programs or functions. If the automating software is poorly developed this may further exemplify the limited flexibility of software automation. Automation software is defined with predetermined rules, in turn poorly defined rule sets can make it difficult for software companies to adapt to changes. Automation has its place in assisting developers in debugging, and assuring the quality of their code, however it is still too elementary to fully replace developers. In order to fully automate the software development process major advancements would need to be made within artificial intelligence as a whole. This uncovers further issues on dependence on technology. As companies further depend on technology, vulnerabilities that cannot be fixed with software can lead to significant losses. Products such as github copilot, Sparx system are best employed as

an assisting tool, not to replace human input. Automation also poses many ethical dilemmas regarding employment. With one third of US jobs at threat due to automation, software testing may be at risk (Manyika 2017). Software testing can be automated, however it requires maintenance as well as overview. Software testing jobs are not currently in an imminent risk due to the need for user input. Many proponents of software testing automation argue that programmers are often unable to perform as complex testing as artificial intelligence. Researchers found that human testers often add unnecessary expenses as well as energy. 80% of software testing is repeating tests, which makes it an ideal task for artificial intelligence (Battina 2019). Software testing still requires user input. Overview of software testing is required to ensure that the automation is thorough. Automation should be used as a safety net and not relied upon, so the software development process is not tainted.

In conclusion, software automation should be automated to some degree. Tools such as the Sparx system allow software to be modeled and documented instantaneously. Code generation promises to increase workflow as well as workload on the programmer. Applications such as chatGPT and Copilot produce quality outputs, as well as analyze code effectively. ChatGPT allows programmers to grasp a more complete understanding of their projects, as well as more effectively fix and replace issues within it. While automation of software engineering poses many issues of feasibility, in most applications artificial intelligence should be used to some degree. Automation should be used in conjunction with a software engineer in order to ensure proper code functionality. Automation within software testing, a major task within the software engineering process, has the possibility of being completely automated. Automation within software engineering is a promising area of research that promises to improve the software development process. Artificial intelligence should be used as a tool to assist programmers, and to expedite the software engineering process.

Work Cited

Alturayeif, Nouf, et al. “An Automated Approach to Aspect-Based Sentiment Analysis of Apps Reviews Using Machine and Deep Learning - Automated Software Engineering.” SpringerLink, Springer US, 9 Sept. 2023, link.springer.com/article/10.1007/s10515-023-00397-7.

Author links open overlay panelArghavan Moradi Dakhel a 1, et al. “GitHub Copilot AI Pair Programmer: Asset or Liability?” Journal of Systems and Software, Elsevier, 2 May 2023, www.sciencedirect.com/science/article/abs/pii/S0164121223001292.

Author links open overlay panelChao Wu, et al. “Residual Attention and Other Aspects Module for Aspect-Based Sentiment Analysis.” Neurocomputing, Elsevier, 13 Jan. 2021, www.sciencedirect.com/science/article/pii/S092523122100028X.

Author links open overlay panelChao Wu, et al. “Residual Attention and Other Aspects Module for Aspect-Based Sentiment Analysis.” Neurocomputing, Elsevier, 13 Jan. 2021, www.sciencedirect.com/science/article/pii/S092523122100028X.

Automating Test Automation | IEEE Conference Publication - IEEE Xplore, ieeexplore.ieee.org/document/6227131. Accessed 5 Oct. 2023.

Automation Testing in Software Organization - Researchgate, www.researchgate.net/profile/Uday-Patkar/publication/362517283_Automation_Testing_In_Software_Organization/links/62edf010505511283e94c732/Automation-Testing-In-Software-Organization.pdf. Accessed 5 Oct. 2023.

Battina, Dhaya Sindhu. “Artificial Intelligence in Software Test Automation: A Systematic Literature Review.” SSRN, 26 Jan. 2022, deliverypdf.ssrn.com/delivery.php?ID=236126126100127092025029122087074126039041069077000060066088087097103121110004109095058000057047061023060103025111065087004113056073082011074084088093081119087025098073054079125067111114003088082100067078088008107082123020023097010082066080006086091089&EXT=pdf&INDEX=TRUE.

“Generate Documentation.” Sparx Systems, sparxsystems.com/enterprise_architect_user_guide/14.0/model_publishing/rtfdialogoptions.html. Accessed 4 Oct. 2023.

Hansson, Emilia, and Oliwer Ellréus. “Code Correctness and Quality in the Era of AI Code Generation: Examining Chatgpt and Github Copilot.” DIVA, 3 July 2023, www.diva-portal.org/smash/record.jsf?dswid=-2009&pid=diva2%3A1764568.

Jiang, Long, et al. “Target-Dependent Twitter Sentiment Classification.” ACL Anthology, aclanthology.org/P11-1016/. Accessed 4 Oct. 2023.

Manyika, James, et al. “Jobs Lost, Jobs Gained: What the Future of Work Will Mean for Jobs, Skills, and Wages.” McKinsey & Company, McKinsey & Company, 28 Nov. 2017, www.mckinsey.com/featured-insights/future-of-work/jobs-lost-jobs-gained-what-the-future-of-work-will-mean-for-jobs-skills-and-wages.

Ritala, Tukka. “Deep Learning Enabled Semantic Communication Systems | IEEE Journals ...” IEEE Xplore, IEEE, 19 Nov. 2007, ieeexplore.ieee.org/abstract/document/9398576.

Shyh-Kwei Chen, et al. “Reversible debugging using program instrumentation.” IEEE Transactions on Software Engineering, vol. 27, no. 8, 2001, pp. 715–727, https://doi.org/10.1109/32.940726.

View of Use Chat GPT to Solve Programming Bugs, journal.hmjournals.com/index.php/IJITC/article/view/1679/1993. Accessed 4 Oct. 2023.