Revista Review Index Journal of Multidisciplinary 2022; 2(3):28-33
eISSN: 2583-0031
Double-blind peer-reviewed | Refereed Quarterly Online Journal

Will Artificial Intelligence become alternative to Software Engineers? - A Futuristic Approach

RESEARCH PAPER

*Nishant R. Mahato

PG Student, Graduate School of Engineering and Technology, GTU, Ahmedabad

Abstract

Investigating what will happen in the field of software engineering in the year 2050 is the purpose of this research. In particular, it draws attention to certain predicted best practises in the industry, the prospective roles of software engineers, and the ways in which artificial intelligence may affect the future of software engineering frameworks and the responsibilities of software engineers. In addition to this, it demonstrates the issues that are occurring right now and provides guidance on how to address them in the future. Finally, it anticipates the challenges that may arise in the future and suggests solutions to circumvent them. These forecasts were arrived at by drawing inferences about the future from existing information drawn from either the past or from current practises and ongoing research in the field of artificial intelligence software. In addition to that, a qualitative approach was used, and a survey was administered online to specialists in the fields of artificial intelligence and software engineering. As an overall outcome of this study, the replies and expectations from a variety of targeted surveys agreed that the future of the software engineering industry would undoubtedly change. On the other hand, software engineers would be the primary forces that may influence this future as well. Those who will either continue to maintain them dominating in this sector or who have been left behind and supplanted by other systems or enterprises.

Keywords: *Software Engineering, Software, Artificial Intelligence, Software Engineer*

Article Publication

Published Online: 30-Sep-2022

***Author's Correspondence**

Nishant R. Mahato

PG Student, Graduate School of Engineering and Technology, GTU, Ahmedabad

nishant.mahatopgit@gmail.com

10.31305/rrijm2022.v02.n03.005

Scan and access Online

Introduction

Any individual is capable of coding, programming, or applications can be created or auto-coded by systems with artificial intelligence (hereinafter referred to as AI). Software engineers will become obsolete or be replaced by AI. These and numerous other predictions were among the news that was circulating. And for each software developer, there are nerve-wracking arguments about what the middle of the 21st century may bring for their profession (here and later). Some of these approaches were predicted on SE based on the natural evolution of the surroundings and influential factors that led to many beginning their work more than a decade ago. This is despite the fact that making such predictions is notoriously difficult. The majority of these approaches were founded on some already-established facts and developments in the fields of SE and AI.

"The use of a systematic, disciplined, and quantified methodology to the creation, operation, and maintenance of software; that is, the application of engineering to software") is how the IEEE Standard Glossary of Software Engineering

Terminology defines software engineering (SE). (IEEE, 1990). On the other hand, artificial intelligence (AI) is described as "the designing and construction of intelligent agents that acquire precepts from the environment and perform actions that impact that environment." (Brewka, 1996). As a result, the distinction between the fields of SE and AI is made abundantly clear by both definitions. SE refers to the practise of adhering to standard methods and frameworks when designing software systems, whereas AI focuses on finding ways to make such systems more livable in more environments. As a human being who demonstrates intelligence and caring in whatever way. At the moment, people are employed in the roles of systems analysts and designers, creators or coders, troubleshooters, and testers. On the other hand, some people believe that artificial intelligence (AI) may one day be able to replace humans in the aforementioned occupations, leaving humans with nothing to do in the future.

Software engineers may further appreciate their significance and worth, become aware of the obstacles that are likely to arise in the future, and make plans to meet those issues head-on in order to protect both their existence and their significance. The remaining portions of the paper are organised as follows: It starts off with a history lesson on computers and software, which helps to establish the methods that are now used in software engineering and identifies the primary issues that are faced in that industry as well as solutions to those challenges. The current situation of the SE domain is once again brought to light and emphasised by this. Regarding the present state of AI development in the direction of automating the majority of the phases of the software development lifecycle framework. The results are presented at the conclusion of the publication, along with suggestions for further research.

Literature Review

History of Computing

It is possible to gain valuable insight into how the future of the current computing age may develop if one begins by illustrating how the features and directions of computing have developed since the invention of the field all the way up until the present day. In the year 1613, the term "computer" had its first appearance in a book titled "The Yong MansGleanings" (R. B., 1614). It was used in this context to refer to a person who was responsible for carrying out calculations and computations (Wikipedia, n.d.). At that time and from that point on, the necessity for these calculations, as well as the human ability to execute more sophisticated ones, became increasingly important. This was especially true when the requirement was heavily required by the requirements of navigation and military demands. Therefore, at the beginning of the 19th century, he conceived of the idea of automating the process of such mathematical calculations, and he invented a difference engine, which accepted a limited number of inputs with certain characteristics and performed only specific mathematical operations could execute, which led to Analytical ideas and designs. It is generally agreed that the computer's engine is one of its most important architectural components.

The earliest computers had continuous programmes, and any change to the machine's design or structure required a change to its function. However, once the software abstraction layer was introduced, the hardware architecture of the machine could be fixed, and any change to the machine could only be implemented at the software layer. Then, as a result of that concept, the development of computer hardware took place over the course of time, and it progressed from having a computer hardware that could run only one task or programme at a time (manually; by physically feeding it into the machine) to having a computer hardware that supports the time sharing model, which enables the user to multitask automatically and simultaneously, thereby increasing the number of programmers and software.

As a result, there was a rise in demand for computers in the 1960s in the fields of business, science, and engineering. In the same decade, there was also a rapid increase in the demand for software programmers to create, correct, optimise, or enhance the millions of computer programmes that were being developed for limited uses in parallel with the development of computer hardware capabilities and facilities. Everyone was coding in a way that was incomprehensible to others, in a way that was more complex, and in a more efficient way that wasn't using hardware level capabilities and in detecting and troubleshooting bugs that used to delay and thus in software creation, which resulted in what was called a software crisis in that era. The software crisis was caused by the fact that everyone was coding in a way that was incomprehensible to others (Pressman, 2009). The issue of the need for solutions to the software problem was brought up for the first time in 1968 at

the NATO Software Engineering Conference, which was also the first time the term "software engineering" was used to refer to the discipline of study.

Some of these demands were eventually met through a number of different initiatives, while others are still being met up to this very day. One of these efforts was to create standard software development lifecycle frameworks or models, which simplified understanding of programme design and made the software development process much simpler, less expensive, faster, reusable, and faster to understand by a variety of developers. Another one of these efforts was to create standard software development lifecycle frameworks or models that simplified understanding of programme design. Detection of errors and various methods of problem solving (Naur and Randall, 1968).

Frameworks for the creation of software continue to undergo change even now. In spite of the fact that these models come in a variety of flavours, they all adhere to a basic structure that includes the following stages: requirements analysis, requirements design, coding or implementation of the design, testing and bug fixing in the coding, moving the product into a production environment, and continuing to maintain it. Lay (Urban, 2015) (Glass, 2002). As a result of the development of that lifecycle, software engineers started doing a variety of tasks and were allocated to a variety of professions, including software analysts, software designers, software developers, software testers, and software support. In addition, each phase has undergone a few advances that have led to improvements, and the following sections will highlight some of these advancements that can assist us in predicting the future of software engineering.

Software Abstraction Layers

Coding the programme is the first and, to this day, the most crucial stage of the software development life cycle. This holds true for each software engineer. The addition of greater abstraction layers to programming languages resulted in an improvement to the previous method of coding. Beginning with a low-level programming language was necessary, such as writing binary machine code. This language was incomprehensible to humans and presented challenges in the areas of coding, comprehending, and debugging. After that, medium level and high level programming languages emerged, such as Assembly, C, C++, and Java, amongst others. These languages made programming languages more human readable and comprehensible by software experts, which facilitated the process of issue discovery and fixing. Additionally, when higher-level languages are utilised, the coding process will need less time since the number of lines of code will be decreased. In addition, new software abstraction layers emerged, which enabled designers to reposition palettes inside a GUI interface by dragging and dropping them, with the result that the code corresponding to the palette was immediately produced. The Visual Programming Language is an additional fascinating abstraction layer that programming offers (VPL). With this particular programming language, users are able to write code visually rather than with text by dragging, dropping, and establishing connections between various graphical objects (Jost, Ketterle, Budd, &Limbach, 2015). The DRAKON programming language is an example of a virtual programming language (VPL). This language was developed by the Russian space programme to make it easier for newcomer developers (Wilson &Oram, 2008) to interpret the complex code that 20,000 programmers had written for the artificial intelligence that controlled the Buran space craft.

A further advancement in the software development life cycle's modelling and designing phase, where programmes such as the Unified Modeling Language (UML) were created to help model or visualise the design of software systems specifically needed by defining the structure of complex programmes Used to do and made it more visible and understandable, particularly for novice programmers who want to start coding with existing complex project code. This is a further development in the modelling and designing phase. It made it easier to develop templates and libraries, which in turn made it simpler to reuse previously developed models and source code in a variety of applications (Booch, Rumbaugh, & Jacobson, 1998).

On the basis of the information presented above, previous researchers hypothesised that the future of software engineering would be characterised by the following characteristics: software systems would be designed at a high level of abstraction, rather than coding programmes line by line, by ordinary people (or any person); software systems would be able to programme, code using existing programmes, self-verify and test; software systems would be agile and fast to meet user requirements; and business or end-user data would be saved regardless of where in the system it was stored. Change

Artificial Intelligence (AI)

When Alan Turing addressed the question "Can computers think?" in his article Computing Machinery and Intelligence, we looked back at several instances in which the same terminology was used to refer to people and computers (machines) (Hodges, 1997). Artificial intelligence (AI) is the study and development of computer systems that mimic human intelligence and behaviour in order to do tasks previously thought impossible (Urban, 2015). The two primary types of AI are described here, along with examples of how each type's present uses and research might aid in managing or enhancing different stages of the software development lifecycle.

Week or Narrow AI

This umbrella term encompasses every type of recent AI advancement. Systems in this class are tailored to a specific use case, and despite their sophistication, they do not display traits of natural intelligence or self-awareness (Dvorsky, 2013). Siri from Apple is an application of Narrow AI.

The Google Self-Driving Car, Watson, or AlphaGo Zero. One such intriguing instance is AlphaGo Zero. To my knowledge, this is the first computer programme to defeat the world champion in Go (China's Game). Professionals and amateurs alike may play the game after receiving different types of training, as well as play against themselves and get insight into how they could better their skills. Although the aforementioned systems are clever in certain respects, they are not shown as possessing human levels of intellect and hence are limited in their ability to act.

Here are some AI systems that aim to replicate the tasks involved in the various stages of software development lifecycles. During the software development testing phase, researchers have developed an AI system that can choose the best trained test cases to run depending on the type of the software fault fixed. The basic concept is to learn which test cases are comparable in disclosing software bugs (Wang, Wu, Lee, & Yao, 2011). Week of the Genetically Engineered System is another another illustration of artificial intelligence. It overcomes the difficulty of auto coding software by drawing inspiration from the natural biological activity of genes to pick a necessary code (based on a knowledge of the issue variable) from a pre-trained library of codes (Spector, 2011).

Strong or General AI

When an AI system reaches or surpasses human intelligence, it is said to be "general AI," since its intelligence may be applied to a wide variety of contexts and problems (Urban, 2015). The average prediction from a study of hundreds of experts on when AI systems might achieve that level of intelligence is the year 2040. (Bostrom, 2014). However, because to this progress, artificial intelligence (AI) will likely replace software engineers (or humans) throughout all stages of the software development life cycle by the year 2050.

Research Method

In the preceding sections, we discussed some of the existing approaches to SE, and we saw how AI may be used to foretell the development of SE. Nonetheless, a qualitative exploration strategy was used to inquire into the future of the SE domain and how AI may anticipate the moulding of that future, with the goal of determining if these expectations are of SE or AI knowledge and exploring further predictions. The questions displayed in the Appendix were written and approved by a professor with extensive experience in software engineering. Most of the people who agreed to take part in this study were thought to be SE and AI professionals working at top-tier, cutting-edge organisations in the UK and UAE. He was also probed on the veracity of the concerns voiced. The subsequent subsections will elaborate on them, as well as the procedure that should be followed.

Participants

Users of LinkedIn, a social media site for professional networking, were the focus of this research. Users of LinkedIn, a social media site for professional networking, were the focus of the study's demographic analysis. The vast majority of these people held positions in medicine, research, or education at prestigious institutions like the Massachusetts Institute of Technology (MIT), Harvard University (HBS), Cambridge University (Cambridge), Google, Microsoft, Apple, Adobe, IBM,

Samsung, Amazon, NASA, Facebook, Uber, Snapchat, LinkedIn, and many more. In just one month throughout this study period, 995 unique users were allowed for connections.

Tools

SurveyMonkey, an online survey platform, was used to create the nine primary survey questions. The first three inquiries are meant to glean information on the respondent's demographic profile, such as their occupation, level of experience, and current employer. The rest of the questions were generally open-ended and were designed to extract participants' perspectives about the future of software engineers in the year 2050, the obstacles they may encounter, and how to overcome such challenges.

Procedures

On the survey's introduction page, we briefly outlined the study's goals. In addition, the fact that respondents would be answering in an anonymous fashion was highlighted in order to encourage candid responses from them. It was asked in the conclusion if such a study is still relevant now and if the research subject and survey questions were valid. LinkedIn members were invited to take the survey through email at irregular intervals over the course of 23 days.

Discussions

Responses to the survey were analysed in a broad sense, without focusing too much on categorising respondents as businesspeople, academics, or researchers. The large discrepancy in response rates between regions made it challenging to do this kind of study. This may be done in the future, when survey requests are sent out to a larger pool of academic and research professionals in order to acquire a more complete picture of the future of each field and make more accurate predictions.

Conclusion

By 2050, software engineering will have seen significant development and change. To make coding easier for everyone, the software development process was likely to take into account new methods, such as new, simpler, and more human-friendly software abstraction layers. As an added bonus, it was planned to have AI systems produce code by exporting or reusing existing code (or lines of code) as needed. Finally, you may always use a different choice as a replacement for one of the others. They are still expected to have a role in the future, either as moderators or creators of AI systems, but only if they stay up with the latest technology, keep researching opportunities, build the necessary skill sets, etc. Utilize AI but don't rely solely on it. Instead, then being used as a standalone SDK.

This project received several useful replies in just a month owing to the time constraint, so additional ideas can be collected as future work. However, it's possible that additional individuals may be asked to provide their thoughts. In addition, other researchers can verify the study's findings and compare them to the experiences of software developers at a given moment, or they can gather further future probabilities, between now and 2050.

References:

[1] Marji, M. (2014). Learn to Program with Scratch: A Visual Introduction to Programming with Games, Art,Science, and Math. California.

[2] Bostrom, N. (2014). Superintelligence: Paths, dangers, strategies. Superintelligence: Paths, Dangers, Strategies. https://doi.org/10.1017/S0031819115000340

[3] Gerard O'Regan. (2012). A Breif History of Computing. Springer Science & Business Media. Ireland. Glass, R. L. (2002). Facts and Fallacies of Software Engineering. October. https://doi.org/http://dx.doi.org/10.1109/FOSE.2007.29

[4] Huws, C. F., & Finnis, J. C. (2017). On computable numbers with an application to the AlanTuringproblem. Artificial Intelligence and Law, 25(2), 181–203. https://doi.org/10.1007/s10506-017-9200-2

[5] Martin Campbell-Kelly. (1988). Charles Babbage's Table of Logarithms (1827). IEEE Annals of the History of Computing., 10(3), 159–169.

[6] Naur, P., & Randell, B. (1968). Software Engineering: Report of a Conference Sponsored by the NATO

[7] Science Committee. NATO Software Engineering Conference, (October 1968), 231. https://doi.org/10.1093/bib/bbp050

[8] Spector, L. (2011). Genetic Programming and Evolvable Machines: Editorial introduction. Genetic Programming and Evolvable Machines, 12(1), 1–2. https://doi.org/10.1007/s10710-010-9127-9

[9] Wang, F., Wu, C. J., Lee, Y. C., & Yao, L. W. (2011). Regression testing of bug-fixes with AI techniques. In Advances in Intelligent and Soft Computing (Vol. 124, pp. 345–354). https://doi.org/10.1007/978-3-642-25658-5_43

[10] Wilson, G., & Oram, A. (2007). Beautiful Code: Leading Programmers Explain How They think.O'Reilly.Sebastopol.CA

Author's Biography:

Nishant R. Mahato *has received his Btech degree from Amity University, Lucknow. Presently he is pusuing Mtech from Graduate School of Engineering and Technology, GTU, Ahmedabad.*

How Cite this article?

Mahato, N. R. (2022). Will Artificial Intelligence become alternative to Software Engineers? - A Futuristic Approach. *Revista Review Index Journal of Multidisciplinary*, *2*(3), 28–33. https://doi.org/10.31305/rrijm2022.v02.n03.005