International Journal of Information Technology and Language Studies (IJITLS)

The Future of Software Engineering by 2050s: Will AI Replace Software Engineers?

Lubna Mahmoud Abu Zohair
Lubna.abuzohair@gmail.com

The British University in Dubai, Dubai, United Arab Emirates

Abstract. This research aims to explore the future of software engineering domain by 2050s. In specific, it points to some foreseeable best practices in the field, prospected roles of software engineers, and how artificial intelligence could shape the future of software engineering framework and engineers' roles. Furthermore, it demonstrates current challenges and how they will be tackled in the future and forecasted future challenges and ways to avoid it from now. These anticipations were determined by inferring the future from some existed facts from history or current practices and researches in the fields of software artificial intelligence. Not only that, a qualitative method was followed, and an online survey was conducted and directed toward software engineers and artificial intelligence professionals. As an overall outcome of this study, various analogical surveys' answers and anticipations agreed that the future of software engineering field would definitely change, however, software engineers will be the main actors who can shape this future in such a way that will either keep them dominant in this field or left behind and replaced by other systems or professions.

Keywords: *Software, Software Engineering, Artificial Intelligence, Software Engineer, AI, Future, Programming, Code, Automation*

1. Introduction

Any person will be able to code, programs or applications may be created or auto-coded by Artificial Intelligence (here and after AI) systems, software engineers will be obsolete or replaced by AIs, and many more other forecasts were among the circulated news and scary debates for any software engineers about how their future (here and after SE) was foreseen during the mid of 21st century. Despite the difficulties of such predictions, most of these outlooks were based on some existing facts and developments in the fields of SE and AI, and few others were predicted based on the natural evolution of the ambient and influential factors on SE that started many decades ago.

SE, as defined by IEEE Standard Glossary of Software Engineering Terminology, is "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software)." (IEEE, 1990). However, AI is defined as "The designing and building of intelligent agents that receive percepts from the environment and take actions that affect that environment." (Brewka, 1996). So, both definitions clearly reflect the distinction between SE and AI fields, as all about SE is to follow standard methods and frameworks while designing software systems, while AI cares more about making such systems more alive and acts smart somehow like a human. Currently, human are the systems analyzers and designers, creators or coders, troubleshooters, and testers. However, it has been argued that in the future AI could pose a threat of replacing the human in the aforementioned roles leaving them doing nothing in the future. Besides the last question, this paper aims to tackle and answer the following research questions as well:

1.1. What will be the state-of-the-art and state-of-the-practice of SE by 2050s?

1.2. How will AI participate in shaping the future of SE?

1.3. What will be the major problems and challenges by then?

1.4. What will be the visionary ideas for overcoming those problems and challenges?

By answering that, software engineers can realize more their importance and value, know the expected future challenges, and be prepared to outface them and preserve their existence and value. The rest of the paper is constructed as follows: it begins with a review of the history of software and computing that structured the present software engineering practices and points to the former main challenges in that field and how they have been tackled. Then, it spotlights on the present state of SE domain and emphasize on the current AI progress toward automating most phases of Software development lifecycle framework. Last, it illustrates the method of the study, a survey sent to different professions in SE and AI fields in high tech companies and institutes thru their LinkedIn profile. After that, it presents the research method findings that answer the aforesaid research questions. Last, the paper ends with concluding the results, including suggested future work.

2. Literature Review

2.1. History of Computing

Starting by showing how computing characteristics and directions had been evolved since its creation until nowadays could give an important indicator of how the future could be shaped for the current computing era. 1613 spotted the first appearance of the word 'computer' in a book called "The Yong Mans Gleanings" (R. B., 1614), in a context to refer to a person who used to execute calculations and computations (Wikipedia, n.d.). Since that time and onward, the need for these computations, and more complex ones for the human ability to perform was accelerated, especially, when that need was highly demanded for navigations and military needs. Charles Babbage, a mathematician who was among the researchers who published mathematical books, like "Table of Logarithms" (Martin Campbell-Kelly, 1988), where a group of researchers had to create it not only one (Subrata Dasgupta, 2014). He experienced the long times spent in maintaining such tables, how expensive it was, and the difficulties in preserving their correctness, especially, during the copying process (which was done by hand) (Gerard O'Regan, 2012). Therefore, early in 19th century, he thought of automating the process of such mathematical calculations, and invented a difference engine, which was accepting limited input with certain characteristics and can execute only specific mathematical tasks, which led to the thoughts and design of the analytical engine that considered the fundamental design concept of the modern computers. That didn't witness the genesis of the computer science field only, but also the science of the artificial (Subrata Dasgupta, 2014).

In the 20th Century, Alan Turing, who was so called "The Father of Modern Computer" made another break point in the history of computing after laying the theoretical base for stored-programs in his 1936 paper (Huws & Finnis, 2017). He designed and constructed a programmable computer, which was named 'Universal Turing Machine. Alan's blue sky's was like of Babbage's one, they both envisioned a machine that can do human way of thinking, as with Babbage that was reflected by building a mechanical machine that by turning its hands alone it will start to crunch the numbers and executions, but with Turing it was through software layer, where he was the first who thought of importance of software as a separate abstraction layer in computing(Copeland, 2004). To clarify it more, early computers had constant programs and changing its function required to change the design and structure of the machine, however, with software abstraction layer machine's hardware architecture became fixed and the change was only applied in the software layer. Then, from that idea, computer hardware evolved over time, and improved from being a computer hardware that can run only one task or program at a time (manually; by physically feeding it into the machine) to a computer hardware that supports time sharing model that enable multitasks to run automatically and simultaneously, thus, leading to increase in the number of programmers and software.

So, during the 1960s, computer demands in business, science, and engineering were increased, and millions of computer programs have to be created, corrected, adapted, or enhanced for limited uses and the need for software programmers was exponentially growing in parallel with the evolving of computer hardware capabilities and features. Everyone was coding in a way that was not understandable by others, in more complex manner, and in none efficient way that was not utilizing from the hardware level capabilities and caused delays in locating bugs and troubleshooting and thus in software creation, resulting to what was named at that era a software crisis (Pressman, 2009). Then, in 1968 and at NATO software engineering conference, where the first appearance of the name of the field of Software Engineering was witnessed. And the need for solutions to solve the software crisis was raised by then. Later, some of these demands were answered by different kind of efforts, and some others still persist until nowadays. One of these efforts was to create standard software development lifecycle frameworks or models that eased the understanding of program designs and made software development process

much easier, understandable by different developers, less expensive, faster, reusable, and fast errors locating and troubleshooting process (Naur & Randell, 1968).

Until today, software development frameworks are still evolving. Despite the variety of these models, they all share the following phases: analysing requirement, designing the needs, coding or implementing the design, testing and fixing coding bugs, moving the product to the production environment and keep maintaining it (Urban, 2015) (Glass, 2002). Due to the emergent of that lifecycle, software engineers started to act in different roles and assigned different professions, like software analyst, software designer, software developers, software testers, and software support. Furthermore, each phase has witnessed certain improvements, and below sections will highlight some of these developments that can help us in forecasting the future of software engineering.

2.2. Software Abstraction Layers

For any software engineers, software coding is until today the leading and most important stage among the software development life cycle. The way we code earlier was improved by the addition of higher abstraction layers of programming languages. Starting earlier by a low-level programming language, like writing binary machine code that was human unreadable language and suffered from difficulties in grasping, coding, and troubleshooting. Then, middle level and higher-level programming languages were appeared, like assembly, C, C++, Java, and others, and make programming languages more human readable and understandable by software professionals and thus facilitate bug locating and fixing process. Moreover, whenever higher languages are used, the coding process will take less time as the line of codes became fewer. Additionally, other software abstraction layers appeared that allow designers to drag or move pallets in the GUI interface and pallet's related code will be written immediately. Another interesting abstraction layer of programming is the Visual Programming Language (VPL). In this programing language, programmers can code visually instead of textually, by dragging, dropping, and connecting graphical elements with each other (Jost, Ketterl, Budde, & Leimbach, 2015). As an example of VPL is the DRAKON programming language that was developed by Russians space program to facilitate the explaining of 20000 programmers' complex codes, created for the AI that controlled the buran space craft, to new coming developers (Wilson & Oram, 2008). Another example is the Scratch VBL program that is designed by MIT to allow students at schools to create programs easily (Marji, 2014).

Another evolution in the modeling and designing part of the software development life cycle, where programs such as Unified Modeling languages (UML) were created to help modeling or visualize the design of the needed software system by defining the structure of the programs especially the complex ones and made it more visible and understandable, especially by those new people who would like to start coding with existing complex projects codes. It also facilitated the building of patterns and libraries to simplify the re-use of created models and codes in different projects (Booch, Rumbaugh, & Jacobson, 1998). There is a known aphorism for David Wheeler, a computer scientist who was awarded the first PhD in computer science in 1951, where he said: "All problems in computer science can be solved by another level of indirection…except for the problem of too many layers of indirections" (Wilson & Oram, 2008). That's true when its clearly shown how the aforementioned programming and modeling abstraction layers can make software development process understandable and more user-friendly, fast and easy to deploy. However, sometimes that is achieved at the expense of adding delays or time overhead, consumption of disk space, and also missing or hiding lower level codes or even hardware concepts from the code developers (Spolsky, 2002). Therefore, the advocates for software process agility advise always to think while creating abstraction layers about its ability to solve what it is designed to solve and how efficient it is utilizing from the available hardware resources (Urban, 2015).

Based on the aforementioned facts, previous researchers envisioned that the future of software engineering would have these natures: software systems will be designed at a higher level of abstraction as a model-driven instead of coding program line by line, ordinary people (or any person) will be able to program, codes will be created using existed programs, self-verify and test itself, agile and fast enough to achieve user requirements, and save business or end-user data no matter what changes are made to the system.

2.3. Artificial Intelligence (AI)

Since the history of computing, we reviewed some areas where same terms were simultaneously used to point for human and computers (machines), and additionally, its proposed by Alan Turing when that "Could a computer think" as a proposed question in his paper Computing Machinery and Intelligence (Hodges, 1997). Mimicking human way of thinking and acting makes up Artificial Intelligence field, as, generally, the main purpose behind AI applications is to create systems that can understand, solve problems and/or functions Intelligently and Independently (Urban, 2015). Below is the explanation of the main two AI categories, pointing in how current applications and researches in each category could play a vital role in handling or improving software development lifecycle phases:

2.3.1. Week or Narrow AI

All current evolutions in AI field are considered under this category. Systems in this category are focused on one task or application, and despite its complication, it's still not featured with natural intelligence, or self-awareness (Dvorsky, 2013). Examples of Narrow AI applications are Apple Siri, Google Self Driving Car, IBM Watson, or AlphaGo Zero. IBM Watson is a question and answer system that is fed with hug kind of knowledge from different resources and trained to answer the natural human question based on the questions that it's been trained with (IBM, 2008). Also, AlphaGo Zero is another interesting example. It's the first computer program that defeat Go (Chines Game) World's Champion. It did that after it is been trained with different ways amateur and professionals played this game, and also it played with itself and from mistakes it learned how to improve itself more and more. But still, all the aforementioned systems are not featured with human-like intelligence and cannot operate or act intelligently in different fields, as they are acting smart in one or specific field only.

Now, below are some examples of AI systems that are designed to imitate the operations of needed in most software development lifecycle phases. Group of researchers made an AI system that can operate in software development testing phase, as such systems can select best-trained test cases to run based on the nature of the resolved software bugs, and the main idea behind it is to learn how to select test cases based on their similarity to the bug-revealing test cases (Wang, Wu, Lee, & Yao, 2011). Genetic programming systems is another example of week AI. It tackles auto-coding software development challenge by selecting a needed code (based on the understanding of the problem variables) from the already trained database of codes using methods similar to natural biological operations of Genes (Spector, 2011). DeepCoders is another AI system that was created by Microsoft and researchers from Cambridge University that is having the ability to write a working program after inspecting huge database of codes, however, currently its limited to write maximum five lines of code but may be improved in the next few years as its informed by the developers (Balog, Gaunt, Brockschmidt, Nowozin, & Tarlow, 2017).

Google AutoML is another Machine Learning system that can create another Machine learning system (Google, 2017). They started with AutoML vision system which allows anyone (even with no ML background) to upload their images in AutoML vision cloud platform. Then, this system will to find patterns in these photos and create custom visioning system models employed for specific business needs. Moreover, they are forecasting to utilize from such AutoML systems for discovering patterns related to speech, translation, video, natural language recognition, etc. Furthermore, by the aim of making AI solutions affordable to anyone, Google make TensorFlow AI library online platform open to the public (Google, 2015). This platform contains free and ready to use open-source machine learning codes, and it allows other to use it as a way to accelerate the improvements on that domain, which could lead to further developments in the aforementioned AI systems that can be used or applied in different phases software development process.

2.3.2. Strong or General AI

General AI is the point when AI systems' intelligence will reach or exceed human-level intelligence where such intelligence can be applied generally and not only into a specific field or use case scenario but multi ones (Urban, 2015). Hundreds of researchers were surveyed to know how soon AI system will reach that level of intelligence and the average believe that this will be seen not before the 2040s (Bostrom, 2014). But, as a consequence of that evolution, we may expect that AI will replace software engineers (or human) by 2050s in the whole phases of the software development framework.

3. Research Method

Previous sections presented some of the known workarounds in the field of SE, and AI helped in foreseeing the future of SE. Nonetheless, to ensure that these expectations are inlined with SE or AI expertise ones, and to explore more forecasts, a qualitative exploration method was chosen to ask them about the future of SE domain and how AI will take part in shaping that future. The questions, shown in the Appendix section, were prepared and reviewed by an expert professor who is a veteran in the field of software engineering. Most of the chosen participants on this study were considered SE and AI experts who were working on outstanding, high-tech, and well-known institutes in the UK and United Arab Emirates. Besides, they were asked if the raised questions were valid or not. The sub-sections below will talk more about them and the followed methodology in details.

3.1. Participants

This study targeted LinkedIn demography, a social media platform that is designed for business and professional networking. Connection requests were sent to more than thousands of software engineers and artificial intelligence professionals from different countries. The majority of those were working as practitioners, researchers, or lecturers at different top giant companies and universities, like MIT, Harvard, Stanford, Cambridge, Google, Microsoft, Apple, Adobe, IBM, Samsung, Amazon, NASA, Facebook, Uber Snapchat, LinkedIn, and many more. The total accepted connections were nine hundred and ninety-five users, in only one month of this research duration.

3.2. Tools

Nine main survey questions were prepared using the SurveyMonkey tool, i.e., an online survey platform. The first three questions designed to extract some demographic details, i.e., the nature of their work, their years of experience, and which company they are working for. The rest were mostly open-ended questions and they aimed to obtain participants' opinions about the future of software engineering discipline and software engineers by 2050, the prospective challenges that they may face by that time, and to know their visionary ideas to solve or overcome these challenges. The appendix shows the survey questions in more details.

3.3. Procedures

A brief overview of the aim of the research was mentioned on the survey page. Moreover, to ensure responder's spontaneity answers, they were noted that they would be answering in anonymous mode. A question about the validity of the research topic and the survey questions was raised at the end to assure that the question design convinced the readers and to know if such study is important nowadays. The survey email invitation was sent to LinkedIn members in different days at different daytime, and the survey link was left open for twenty-three days.

4. Findings and Discussions

In twenty-two days' survey duration, eighty-one responses were successfully collected among nine hundred and ninety-five LinkedIn connections. Among them, the answers were filtered, and only Fifty-Two responses were considered, and the rest of the responses were excluded since the core research questions were skipped or left unanswered. Fifty-Two completed answers from experts in one-month research duration considered very credible number. All of them agreed about the validity of the research questions, as shown in the answer to questionnaire question 10 in the Appendix. Figure 1, Figure 2, and Figure3 illustrate participants answers to the demographic questions about their workplace, professions and years of experience, in respectively.

As seen from Figure 2, most of the responders were practitioners, and they were working in high tech. companies. The invitation requests targeted those who had AI or SE professions in their job profile. LinkedIn is a business and employment service, and its main use is to network between companies and job seekers (Lemann, 2015), and since the number of companies that are joining that platform is more

than academic institutes, the ratio of the number of practitioners compared to academia or researcher population is more. Besides, the connection requests were highly accepted by those who were working in high tech companies. Regarding participants' years of experience, the related question (Question 1 in Appendix) provided a reasonable range of years starting from zero to fifty, where zero professional are those who didn't complete their first year of experience and the fifty was the reasonable higher limit of years of experience since that was almost the age of SE and AI fields existence. Two years of experience ranges were created, the first included the juniors in SE or AI fields who has less than ten years of experience, and the other year range included the experts or veteran consultants in that fields.

Starting by the core questions, when the participants were questioned about their forecasts for the future of software engineering, i.e., the fourth question in, illustrated in Figure 13 in the Appendix section, they were provided two ways of answering that question. The first was either to pick one or multi options from the suggested forecasts, and the other was to allow them to add other foreseeable visions in the text box. Figure 4-a shows the number of responses for the suggested list of forecasts, and the three most interesting common envisions were: all of them (considering their different years of experience in the field) agreed that the software engineering field would not maintain its current state and it will definitely change, any person will be able to code, which is a strong reflection of (Marji, 2014) point of view, and existing codes and models will be reused somehow in the future.

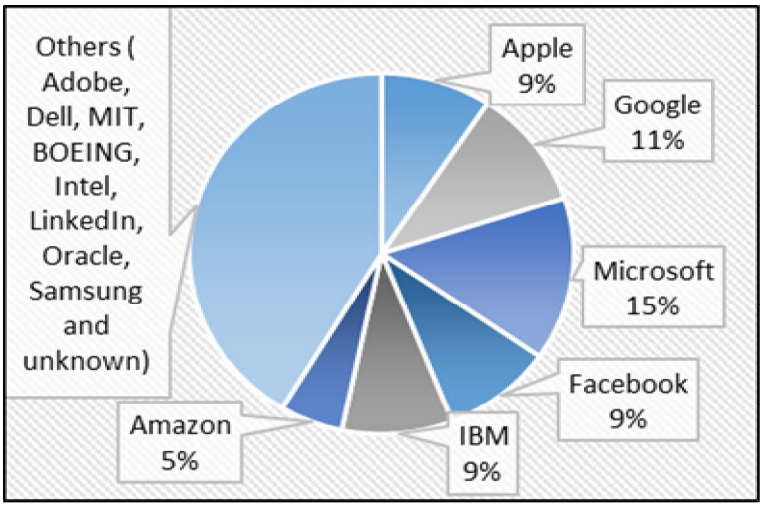

Figure 1. Responders rate per workplace, noting that a responder may have experienced to work in multi workspace.

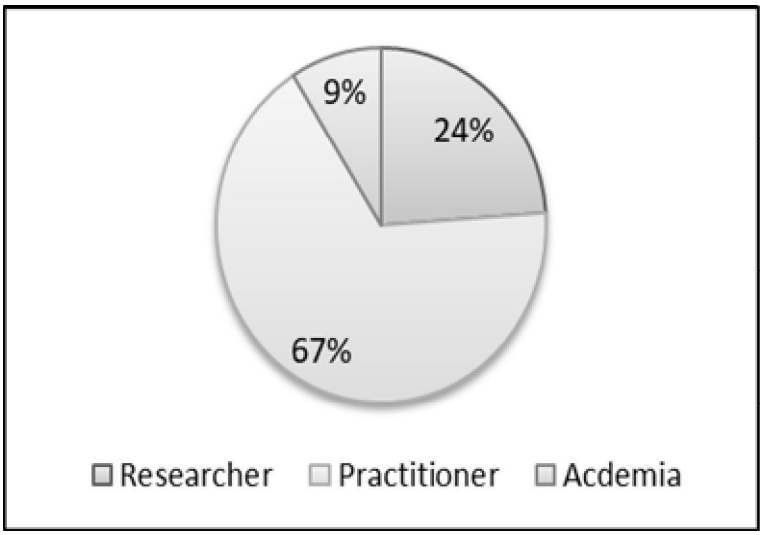

Figure 2. Participants ratio per type of professions. Noting that there are some participants who were working in two or that three fields at the same time.

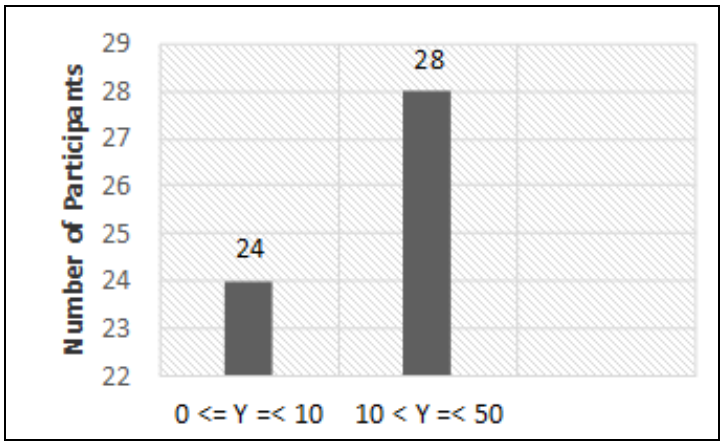

Figure 3. Number of participants per different ranges of years of experience (Y).

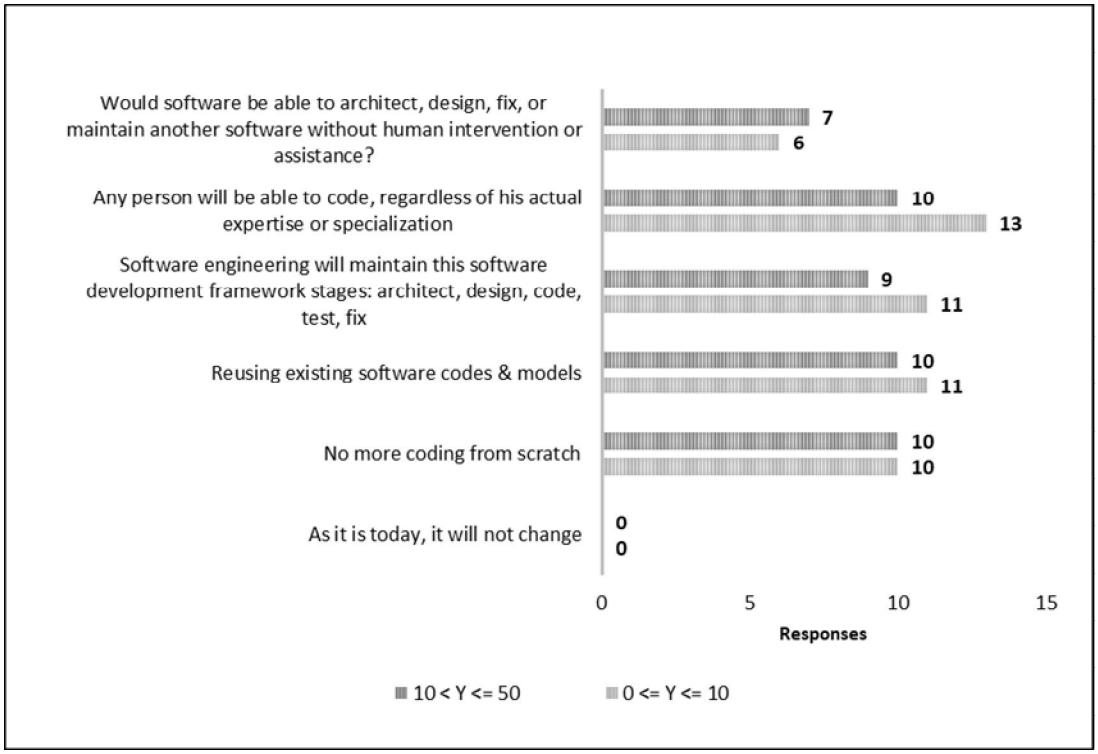

Figure 4-a. Number of responses per suggested forecasts for the future of software engineering by 2050s.

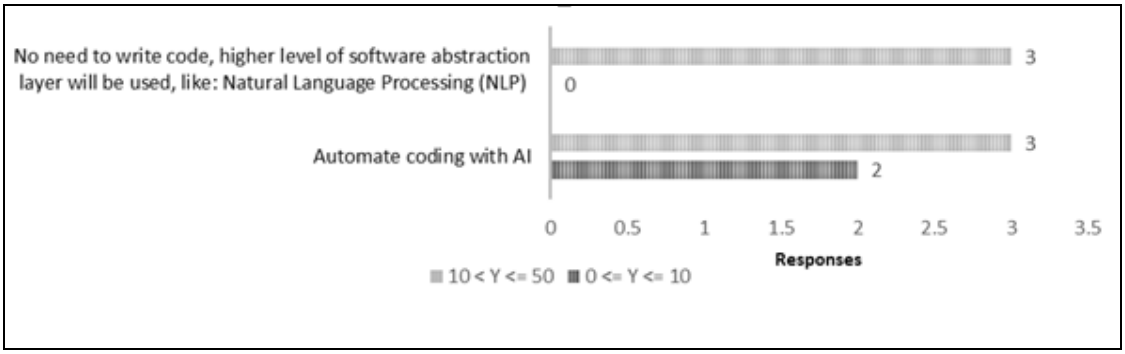

Figure 4-b. Responders forecasts about 2050s future of software engineering field.

However, not all of them added their own forecasts, but among only Fourteen responses to that, the most two repeated answers, as shown in Figure 4-b, were the usage of AI for coding and the usage of higher software abstraction layer for coding, like Natural Language Processing (NLP) instead of typing the code in standard way. Regarding code automation fact, that was consistent with the facts given in sections II, III and IV about the current studies in AI to automate the coding process, such as Genetic Programming, DeepCoder, and AutoML (Google, 2017). Nonetheless, current techniques still have limitations and not considered in the coding process yet, but in the 2050s these limitations might be narrowed, and the usage of these techniques and more advanced ones could be more efficient. As observed in the aforementioned question, forecasts were harmonized and common in the different ranges of years of experience. So, in view of this, and because of the small number of responses that were contributed in the rest of the survey questions, the analysis will be conducted in general by excluding the interpretation of the responses per different ranges of years of experience and make the analysis more general.

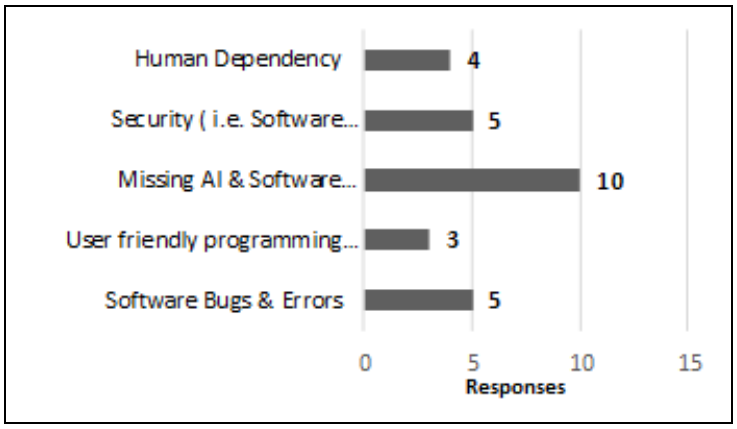

Figure 5. Current challenges in software engineering field that will be tackled by 2050s.

The next question was an open-ended question about the current challenges in software engineering field that will be solved by 2050s, and the most common answers, as seen in Figure 5, were directed toward the need for the automation in the software development process because of three main reasons, one of them was that AIs or systems make less mistakes than human, the other was that the software creation process will be much faster if it done by machines, last systems are much reliable and create less mistakes compared to human. That was also the reason behind considering Human Dependency as one of the existed challenges that should be tackled in the future. Then, when members were questioned about how AI will affect the future of software engineering field and engineers their responses, as displayed in Figure 6, pointed out that AI will never replace Software Engineers, contradicting by that with (Urban, 2015) fact about the strong AI advent, but at the same time they agreed that it would create code and replace coders.

The next question will help in clearing the confusion of the last aforementioned answers. The next question was about the future role of software engineers, and the most repeated answer, as shown in Figure 7, was that they would maintain, moderate, lead, or develop AI systems. So, linking this answer with the previous question responses, it could be expected that AI code and replace the engineers in that role only, but in contrast, the software engineers (with developed expertise) will still be needed and required to maintain, train, moderate, or direct such AI systems to the proper way to create such programs. Additionally, experts assumed that the software engineers' role would maintain as it is, and that can be clearly explained from their answers to the last open-ended question about the main expected challenges by 2050s and how it can be tackled from now. The most repeated answer to that question was that software development process would never take place fully by AI as software engineers will restrict AI usage and only use it as a tool or complementary appliance in the software creation process. That's why there were high assumptions by experts that the role of software engineers during 2050s, in question eight in Figure 17 in the Appendix section, will be as is nowadays.

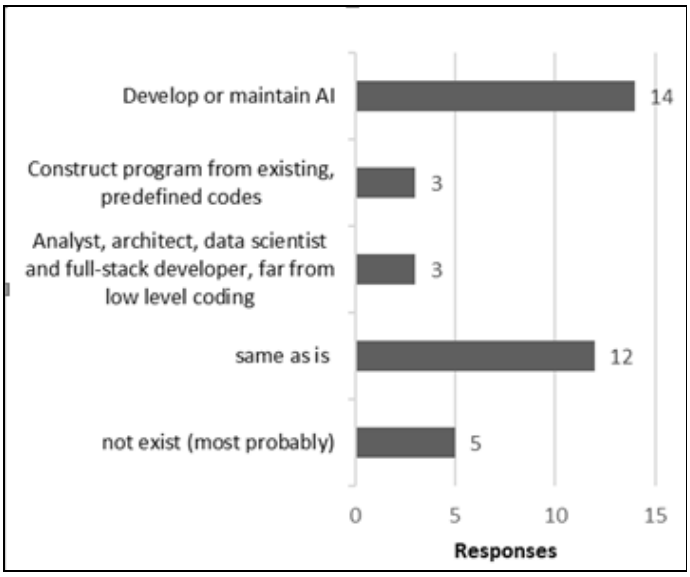

Figure 6. How AI will shape the future of software engineering field and engineers during 2050s.

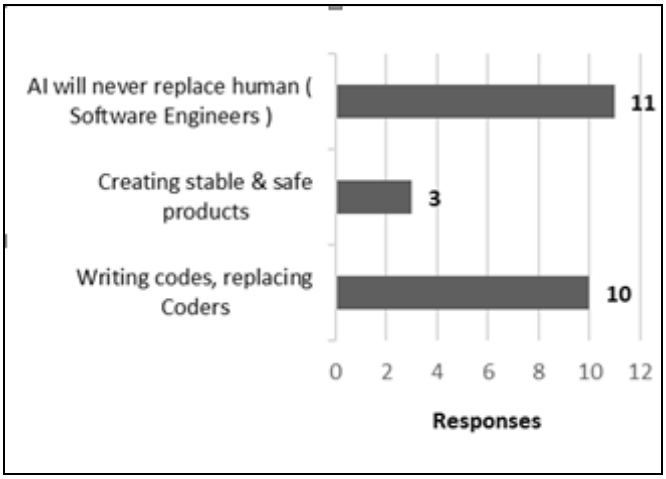

Figure 7. Prospected Role of software engineers by 2050s.

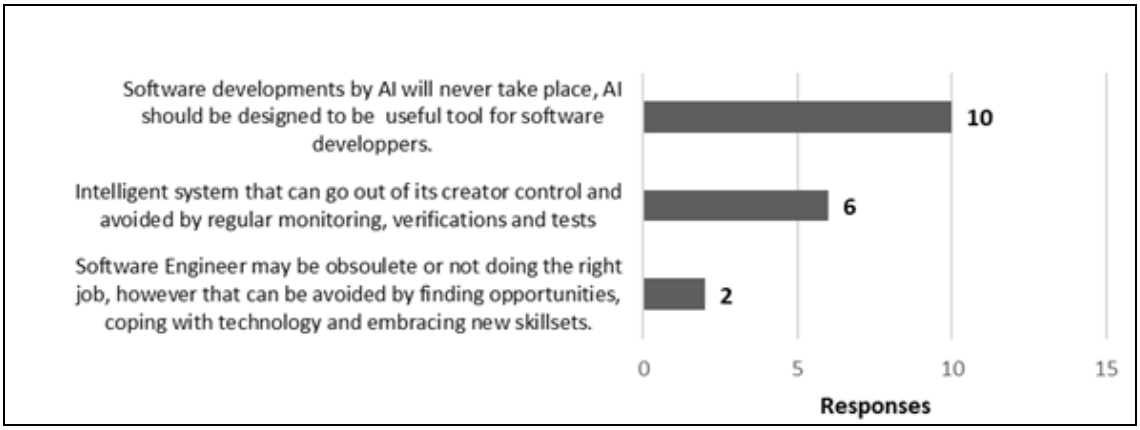

Figure 8. 2050s Envisioned challenges in software engineering field and suggested ways to tackle it from now.

Moreover, another two most foreseeable challenges, as illustrated in Figure 8, were that AI systems could go out of its creators' control and the advised way to tackle that was by keeping such systems in

continuance monitoring, testing, and verification process even during its usage, and for worst situation, precautions should be planned ahead in case of the occurrence of such situations. The other prospected challenge was that the software engineers might be obsolete, and as per responders' responses that such a fate be controlled by the software engineers themselves, such that if they keep developing themselves and coping with the recent technologies they will maintain their value and role despite any challenges will threaten that in the future.

The survey answers were analyzed generally without being more specific in the analysis per different participants' professions responses, i.e., Practitioner, Academia or Researcher. Such an analysis was difficult to be achieved due to the huge variance in the number of responders per each field. However, that could be carried out to the future work, where survey requests will be targeting more academia and researcher participants until enough sample is gathered in each field sufficient for detailed analysis of each to achieve broad future outlook and get more reliable forecasts.

5. Conclusion

The future of software engineering field will change and evolve by 2050s. New methods were prospected to be considered in the software development process, like new, easy and human-friendly software abstractions layers, which could make the coding process an easy job for any person. Besides, AI systems were envisioned to be used for creating codes by exporting or reusing already existed codes (or lines of codes) based on user requirements. Final tip, and as a way to reassure software engineers, no one can dispose or replace them unless they choose to be so. It's predicted that they'll still find a place in the future, either as AI systems' moderators or developers, but only if they cope with the recent technologies, keep searching for opportunities, embracing the needed skill sets and utilizing from AI as complementary tools rather than as independent software development systems.

As a future work, further views could be gathered, as in this project due to time limitation only some valuable responses were received in only one month. However, more could be invited to participate with their views. Additionally, other researchers from now and until 2050 could keep validating the result of the study and compare the findings of this research with the certain state of software engineers at a certain point of time, or they can gather additional future prospects.

References

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2017). DeepCoder: Learning to Write Programm. *Iclr*.

Booch, G., Rumbaugh, J., & Jacobson, I. (1998). *The Unified Modeling Language User Guide*. *Techniques* (Vol. 3).

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

Brewka, G. (1996). Artificial intelligence—a modern approach by Stuart Russell and Peter Norvig, Prentice Hall. Series in Artificial Intelligence, Englewood Cliffs, NJ. *The Knowledge Engineering Review*, *11*(01), 78. https://doi.org/10.1017/S0269888900007724

Copeland, B. J. (2004). Colossus: Its origins and originators. *IEEE Annals of the History of Computing*, *26*(4), 38–45. https://doi.org/10.1109/MAHC.2004.26

Dvorsky, G. (2013). How Much Longer Before Our First AI Catastrophe?

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

Google. (2015). TensorFlow Open Source Machine Learning Framework.

Google. (2017). Cloud AutoML. Retrieved from: https://cloud.google.com/automl/

Hodges, A. (1997). Alan Turing Scrapbook - Turing Test. Retrieved from: http://hps.elte.hu/~gk/Loebner/scraptes.htm

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

IBM. (2008). IBM Watson. Retrived from: https://www.ibm.com/watson/

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). Los Alamitos. *CA: IEEE Computer Society*, *610.12-199*. https://doi.org/10.1109/IEEESTD.1990.101064

Jost, B., Ketterl, M., Budde, R., & Leimbach, T. (2015). Graphical programming environments for

educational Robots: Open Roberta - Yet another one? In *Proceedings - 2014 IEEE International Symposium on Multimedia, ISM 2014* (pp. 381–386). https://doi.org/10.1109/ISM.2014.24

Lemann, N. (2015). The Network Man.The New Yorker.

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

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

Naur, P., & Randell, B. (1968). Software Engineering: Report of a Conference Sponsored by the NATO Science Committee. *NATO Software Engineering Conference*, (October 1968), 231. https://doi.org/10.1093/bib/bbp050

Pressman, R. S. (2009). *Software Engineering A Practitioner's Approach 7th Ed - Roger S. Pressman*. *Software Engineering A Practitioner's Approach 7th Ed - Roger S. Pressman*. https://doi.org/10.1017/CBO9781107415324.004

R. B., G. (1614). Computer. Retrieved from https://en.wikipedia.org/wiki/Human_computer#cite_ref-1

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

Spolsky, J. (2002). The Law of Leaky Abstractions. Retrieved from: https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/

Subrata Dasgupta. (2014). *It Began with Babbage: The Genesis of Computer Science*.

Urban, T. (2015). The AI Revolution: The Road to Superintelligence.

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

Wikipedia. (n.d.). Computer.

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

Appendix

The following are the series of screenshots for the survey's questions, where each figure explains the type of questions and the way of answering it.

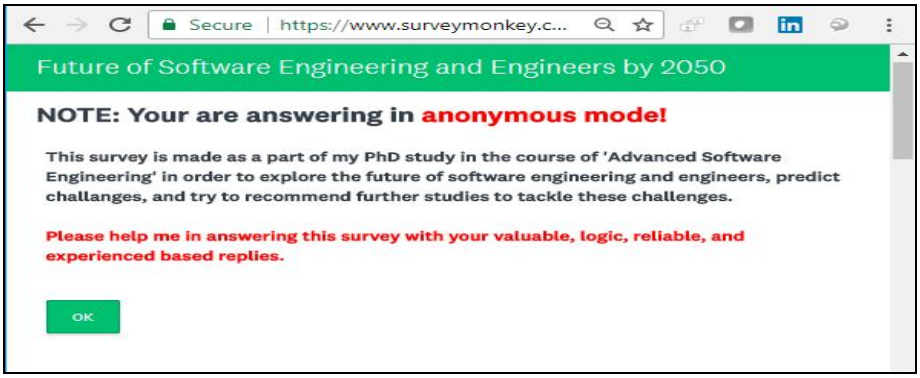

Figure 9. Brief description of the project.

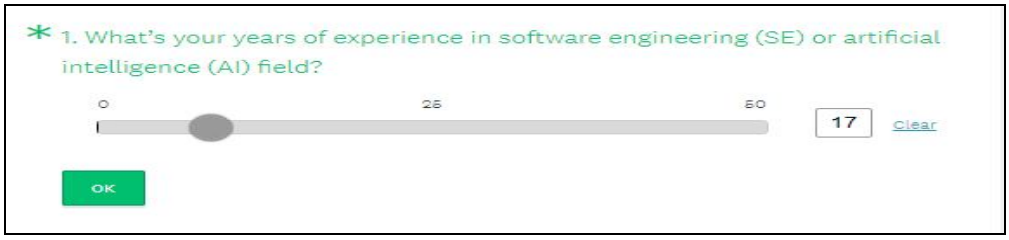

Figure 10. Survey question number 1, slider bar to choose years of experience from 0 to 50.

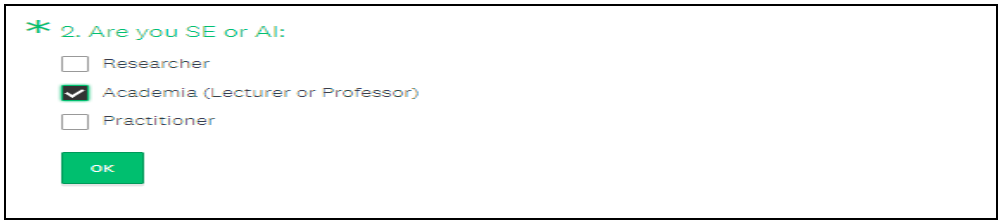

Figure 11. Question 2, checkboxes way of answer to choose one or multi answers at the same time.

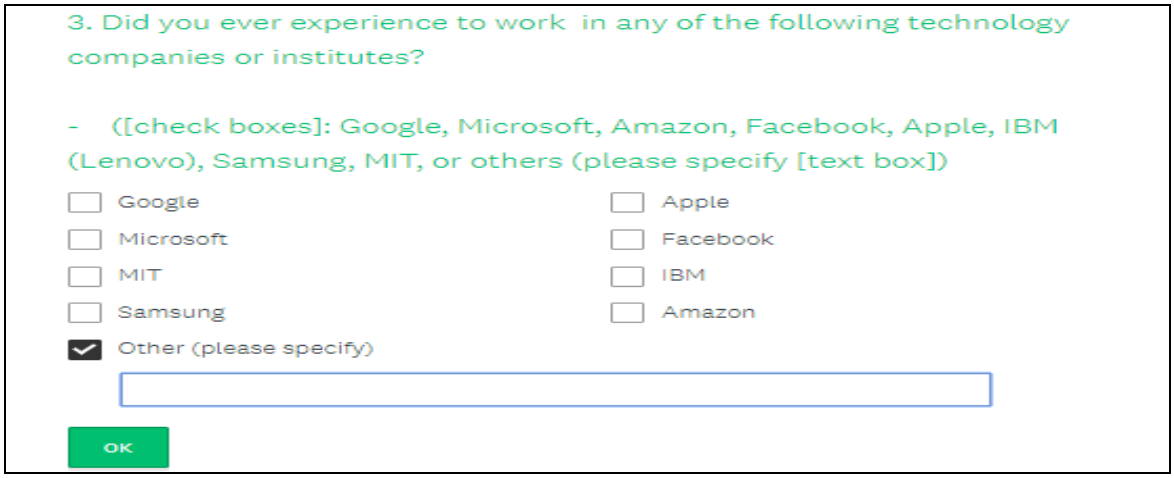

Figure 12. Question 3 shows 2 ways of answers, checkboxes and a text box for other unmentioned high-tech companies.

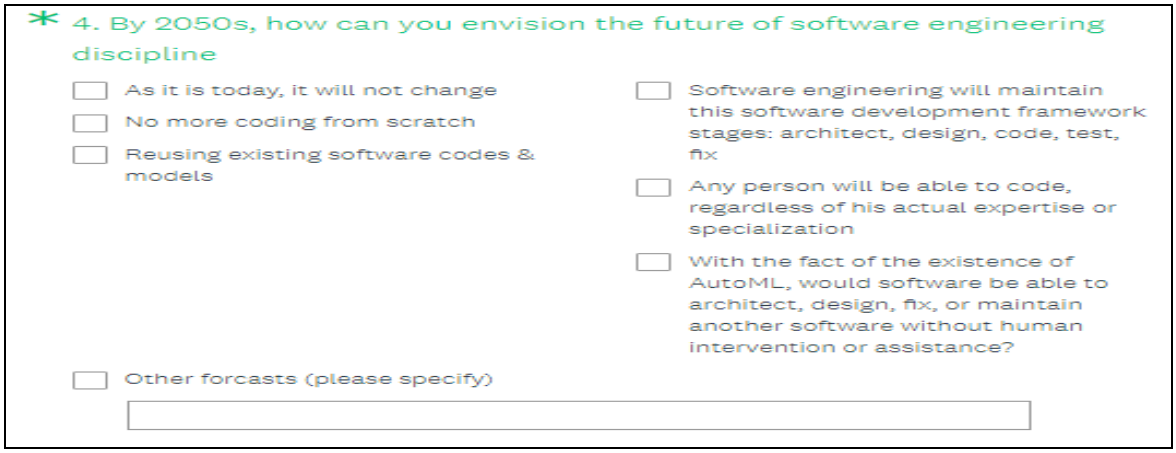

Figure 13. Question 4 shows 2 ways of answers, checkboxes and a text box for other forecasts.

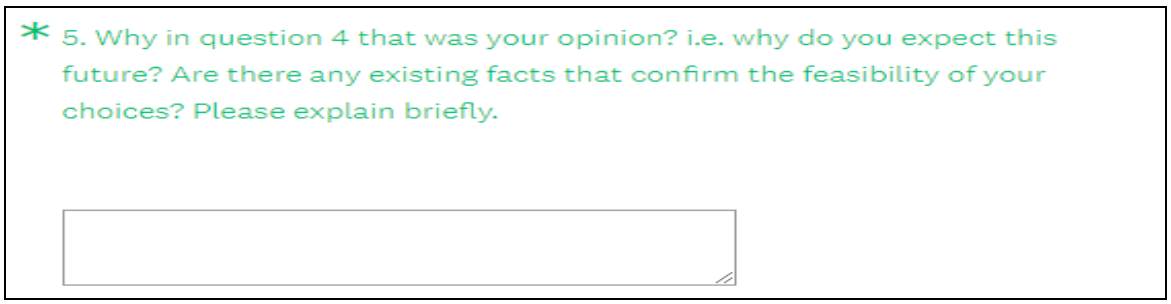

Figure 14. Question 5, an open-ended question.

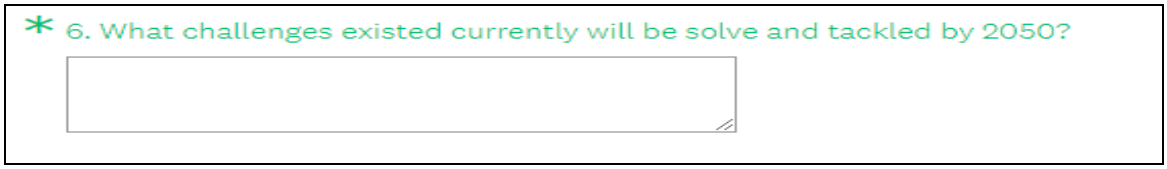

Figure 15. Question 6, an open-ended question.

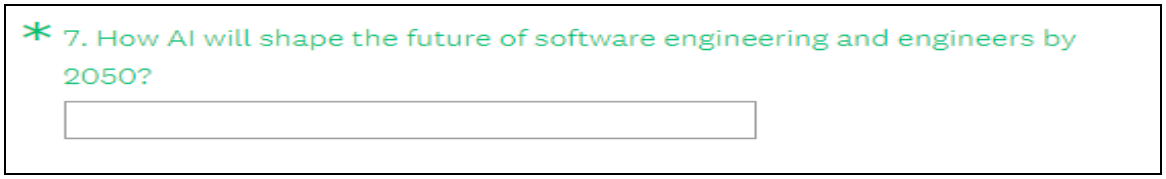

Figure 16. Question 7, an open-ended question.

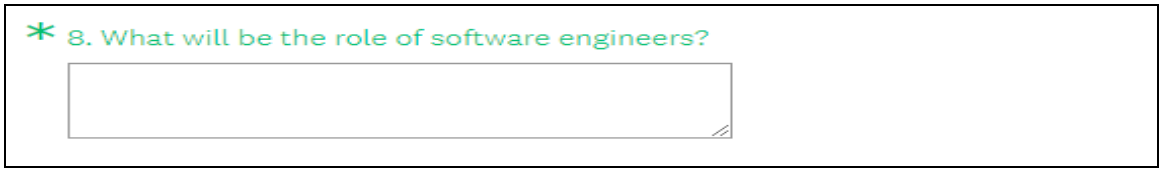

Figure 17. Question 8, an open-ended question.

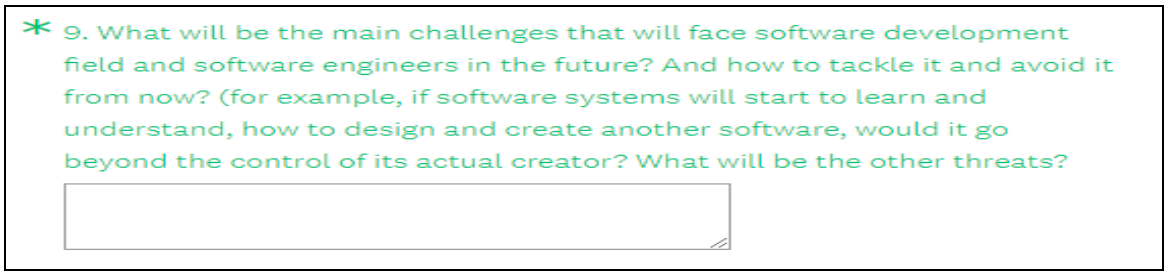

Figure 18. Question 9, an open-ended question.

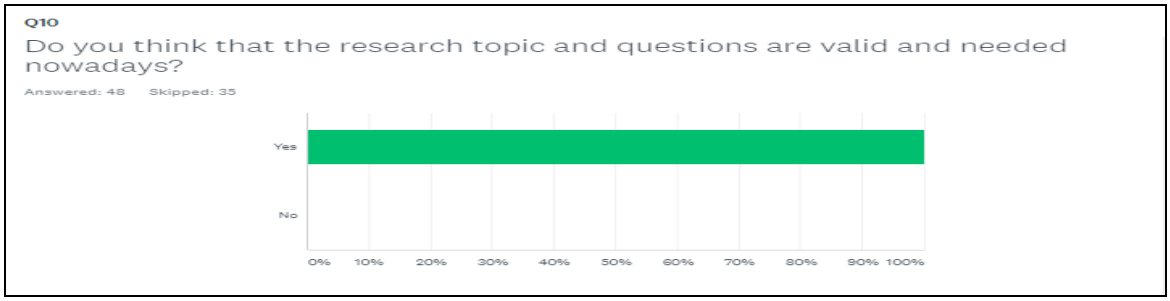

Figure 19. Question 10, Boolean question.