The Past, Present, and Future of Software Engineering

Group #13

Stefan Bukarica, Samson Goodenough, Chen Jin, Jordan Dubrick

Dr. Shaun Gao

Wilfrid Laurier University

190563930, 190723380, 170631720, 201859360

December 9, 2021

Table of Contents

1. Introduction

Software Engineering is the application of engineering principles to create and develop software products for computer systems. Although it is relatively new, compared to other engineering disciplines, it has quickly become one of the most in demand due to the importance software products have in our daily lives. Almost every person, business, and service relies on quality software in order to conduct many tasks. Since the growth of popularity of software products has ballooned in a constantly evolving technological space, there are many reasons to believe it will continue to expand as it has in the past 50 years. With the expansion, it will likely change what the field encompasses along with its customers and consumer products. In this paper, the goal is to identify past trends in the industry of Software Engineering, the current practices, and how these along with other factors will influence the future of Software Engineering. Ultimately, the outcome of this paper is to identify how the past influences the future of Software Engineering, along with how unexpected factors might influence the future.

2. Past of Software Engineering

The need for computers and computing power has existed for thousands of years. Any tool used to calculate, store, record, and communicate data can be seen as a form of a computer. However, the evolution of how we perceive and process data through computers did not ramp up until the mid 20th century. This era brought with it the advent of digital forms of computing, and in turn, software and the disciplines required to build it. Though no one can for sure pinpoint the exact birth of Software Engineering, evidence points to the term first being coined in around 1963-64 by American computer scientist and NASA engineer Margaret Hamilton during her time working on the SAGE project (Booch, 2018).

The Cold War was a catalyst for massive developments in weapons as well as computing technology, as Eastern and Western superpowers poured massive funding into the tech field in an attempt to stay one step ahead of their counterparts. Thus, in the early 1950s, the United States created the SAGE project. SAGE (or Semi-Automatic Ground Environment) which was the largest computer ever built, was an air-defense system which produced images of airspace based on radar data. Built with over 500,000 assembly statements, the sheer scale of the SAGE project not only brought legitimacy to Software Engineering and demonstrated its usefulness, but also laid the foundations for every major software endeavor after it (Jones, 2014). In 1969, the SAGE project was referenced in “Software Engineering Techniques”, a software document created during the NATO’s keystone Software Engineering conferences (NATO, 1969).

The 1960s were a time of innovation in the field of Software Engineering, but these rapid advancements also brought with them a slew of problems. Thus, this era also marked the beginning of the software crisis, which lingered into the 1980s. The software crisis was thought to have begun in around 1965, and was caused mainly by the fact that advances in software were occurring faster than engineers could keep up with projects. In turn, many software projects failed on multiple different levels, from failing to meet requirements, to causing critical errors during user operation, sometimes resulting in injury or even death (Jones, 2014).

The Therac-25 incidents proved to be one of the most impactful, as well as tragic accidents, which occurred at the tail end of the software crisis. Therac-25 was a radiation therapy machine created in 1982. Unlike it's previous iterations, Therac-6 and Therac-20, Therac-25 used a computer to conduct routine operations. Therac-25 had unreliable software locks, which caused the machine to administer dangerous doses of radiation to its patients, killing four and seriously injuring two (Leveson, 2017). These software issues stemmed from lack of proper software testing, sloppy software engineering as well as a lack of communication between engineers and users. Though the Therac-25 incident was very tragic, it led to the creation of the IEC 62304, a landmark software standard used to ensure the proper development and maintenance of software lifecycles.

In 1968 to 1969, NATO held two conferences on Software Engineering in Germany. This was the first time the term "Software Engineering" was brought to prominence. The purpose of these conferences were to highlight issues in Software

Engineering during the software crisis, and to formulate solutions to solve these issues. In order to address these issues, some sort of standardized “rulebook” needed to be developed. Ultimately two documents were produced from these conferences, titled “Software Engineering” and “Software Engineering Techniques" which acted as some of the first guidelines on software development and the best standard practices to go about it. The first document covered topics such as design, production, and service (NATO, 1968), while main sections of the second document covered software specification, software quality, and software flexibility (NATO, 1969). Despite these conferences, the software crisis lasted nearly two more decades until the mid 1980s. However, these conferences still proved to have taken effective steps in order to resolve many major issues in Software Engineering.

By the 1970s Software Engineering was looking to become the fastest growing industry in history. Such growth came hand in hand with the rise of commercial computers and software companies, many of which are now household names, such as Apple and Microsoft (Jones, 2014). The programming languages C and PASCAL were also introduced during this time, which led to massive shifts and improvements in the software field. This included the introduction of new concepts such as middleware, embedded software, databases, project management, etc. Ultimately, all of this brought massive growth to the tech industry and acted as a foundation for the golden age of software and technology (Jones, 2014).

The 1980s marked the age of personal computers. Companies were creating computers which appealed to a much broader audience, and with this came a major software boom. Commercial software and operating systems demand skyrocketed, and the concept of owning a computer was starting to become more mainstream. Additionally, the way in which engineers wrote software was becoming more standardized, and the software boom brought with it many technologies and methodologies (such as object-oriented and structured programming) which made software development more streamlined and easier. By the second half of the 1980s, these new innovations helped bring an end to the software crisis (Jones, 2014).

The 1990s marked possibly the largest shift in the tech industry with the advent of the internet. A vast array of new global markets were introduced along with the internet, and entrepreneurs looked to invest and take advantage of this new platform. This not only increased demand for software engineers, but also brought prominence to new dimensions in the software field, with disciplines such as networking, web design, and security. Many jobs in the software industry also shifted towards being more centered around the maintenance of aging software, many of which were developed during the software crisis (Jones, 2014).

3. Current Day - Present of Software Engineering

The current climate of Software Engineering revolves around a data-centric approach to many tasks. The abundance of data collection and increase in computing power in the 2010's has resulted in many different products becoming necessary in the market, ones that software engineers are required to develop. With the advancements of data collection and processing power, comes the advancement of artificial intelligence. Sourcing mass amounts of data in any field has allowed AI to advance and continue to do so within the past 5 years, and foreseeable future. With the advancements in AI comes many questions regarding Software Engineering principles and processes. Rather than programming software, we can teach pre-existing software to create/perform as we wish. Furthermore, this changes aspects in testing, delivery time, and project management.

The COVID-19 pandemic has not caused much of a change for Software Engineering, since this is a field where most of the work could be done remotely before the pandemic. However, it has changed the remote setting for many other professions that software engineers interact with. Since software engineers work hand in hand with businesses to create software products for clients, the other portion of the field that software engineers interact with has been forced to be remote, which can be both good and bad. The positive side of things is businesses likely save money being online as a result of not operating out of a physical office, therefore the demand for high-quality, more expensive software has increased. The negative side is that the integration has not always been the smoothest, causing gaps in time-management and presentation.

4. Trends in Software Engineering

Software Engineering differs from all other disciplines under the engineering umbrella as it is currently a much more fluid and non-standardized process. However, there are still noticeable trends that most involved in the Software Engineering space will follow to ensure some standardization. As the world of Software Engineering continues to evolve, so do the trends that are associated with it. As of late, the major trends are leaning more towards development methodologies and practices, such as Agile Software Development, where 79.4% of Software Engineering Education studies are mentioning this practice in some way (Cico et al., 2020). Historically it was noted that a need for agility and adaptation will be required in Software Engineering as we progressed into the 2000s, and with the rise of Agile as a popular choice, we are seeing this prediction come to fruition (Cico et al., 2020). Along with the rise of adaptive development methodologies, we are seeing an increase in educators implementing strategies such as Agile, Kanban, and Scrum into their curriculum to further educate future software engineers (Cico et al., 2020).

Furthermore, a major trend in Software Engineering is the realization that the development of software is not a one and done scenario, rather, it is an iterative process (Bogost, 2018). When software was in its early stages and relatively new to the world, they coined Software Engineering as a hope to relate this process to the other engineering disciplines, where once a product was created, it would be difficult to alter. The acceptance of software development being more iterative has led to the trend of increased informality in Software Engineering, allowing the requirement to have a product work on the first try no longer an issue (Bogost, 2018). Currently, the trend that you do not need to attend higher education to enjoy a successful career in Software Engineering, was created because of this increased informality in the industry. As noted by the popular software engineering forum Stack Overflow in their 2020 survey, only 9.7% of respondents noted that they believed higher education was required to become a software engineer (Columbia University, 2021). This is further reinforced by major technology giants, such as Google and Apple, waiving the requirement to attend formal education before becoming qualified.

Regardless of the shift away from requiring formal education to become a software engineer, the class sizes of computer science and computing degrees have continued to grow, and from 2009 to 2015, the number of bachelor's degrees received has risen 74% (Hambrusch, 2018). It is key to note, that even as computer science grows in popularity, and an increasing number of developers are learning through alternate means, there is still a shortage in labor. It is estimated that in the United States alone there will be upwards of 400,000 new graduates in the Software Engineering space, and the industry itself is predicted to need to fill at least 1.4 million development jobs (Columbia University, 2021).

Finally, one of the most notable trends that has become popular in the present day is that of artificial intelligence within Software Engineering. It is commonly being used to replicate the intelligent behaviour of well written software (Harman, 2012). Artificial intelligence is a large space and can contain anything from machine learning, to image recognition, to content extraction. It is becoming increasingly incorporated into Software Engineering and with the release of Github Copilot, it is allowing software engineers to increase the amount of work done daily (Sobania et al, 2021). Artificial intelligence is becoming so readily accessible to software engineers that it is simply being added as plugins to commonly used integrated development environments. One of the main uses of artificial intelligence is to predict what is required, and if the prediction was incorrect, the program should be able to learn and increase its future accuracy. As artificial intelligence is continually incorporated into Software Engineering, it is improving the standardization of software and increasing the accuracy of all software it deals with. This is largely attributed to the fact that artificial intelligence will not make the same mistakes twice, whereas there is always the possibility of human error.

5. The Future Of Software Engineering

With the current practices being discussed, we expect a continuation of the big data and automated problems to continue to be important in the near future as they are relatively new subjects. Through the years, Software Engineering has evolved to producing software products that interact over many different interfaces, the reality is that software engineers are expected to make products that interact over a large system. Looking towards the future, we expect that these systems will increase in size as time goes on. However, with many other advancements such as quantum computing and virtual reality, we also expect the software development life cycle to drastically change for software engineers. With these advancements, we expect software engineers' work to be more abstract and focus mainly only on high-level design, leaving low-level design for existing automated systems.

Quantum computing and more importantly, quantum technology, is rapidly changing the world. Quantum computing is based on quantum mechanics like superposition and entanglement, they are expected to allow computational power to increase exponentially (Piattini et al, 2020). With this, we expect projects to become more complex and ambitious, with large processing power being necessary in fields such as finance or scientific research. With this comes the need to be able to deliver quality quantum software with Software Engineering principles such as quality, delivery, and project management (Piattini et al, 2020).

Another promising development for future software engineers is the creation of a metaverse, a virtual reality environment where humans will be able to spend time with the help of virtual and augmented reality tools. The creation of the metaverse is currently underway, but with its creation, it will change a lot of the ways software engineers work. As more people adopt a metaverse lifestyle, communication, interactions, and resources become more readily available for software engineers, easing some of the processes of the SDLC that require vast amounts of coordination between many entities such as team developers, testers, and businesses.

Finally, the continued development of artificial intelligence will most likely be the biggest driver of change in the industry of Software Engineering. As autonomy increases in many other disciplines, it will as well increase in software development as AI becomes more readily available and developed. For example, allowing AI autonomous control over coding and testing will ease much of the low-level design software engineers have to consider. The artificial labour produced from AI's will allow software engineers to teach existing AI how to perform tasks they need done, this results in producing adaptive control of the environment, or redesigning the world to avoid current problems and create new opportunities (Nanz, 2011). This will make software engineers work revolve mostly around the high-level elements of the SDLC, resulting in a close partnership between AI and engineer, to make these more complex software products feasible for the future.

6. References

G. Booch, "The History of Software Engineering," in IEEE Software, vol. 35, no. 5, pp. 108-114, September/October 2018, doi: 10.1109/MS.2018.3571234.

Nanz, S. (2011). The Future of Software Engineering. Springer Berlin Heidelberg.

Harman, Mansouri, S. A., & Zhang, Y. (2012). Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys, 45(1), 1–61. https://doi.org/10.1145/2379776.2379787

Cico, Jaccheri, L., Nguyen-Duc, A., & Zhang, H. (2021). Exploring the intersection between software industry and Software Engineering education - A systematic mapping of Software Engineering Trends. *The Journal of Systems and Software*, *172*, 110736–. https://doi.org/10.1016/j.jss.2020.110736

Cico, O., Jaccheri, L., Nguyen-Duc, A., & Zhang, H. (2020). Exploring the intersection between software industry and Software Engineering education - A systematic mapping of Software Engineering Trends. *Journal of Systems and Software*, *172*.

Bogost, I. (2018, September 27). *Programmers: Stop calling yourselves engineers*. The Atlantic. Retrieved November 25, 2021, from https://www.theatlantic.com/technology/archive/2015/11/programmers-should-not-call-themselves-engineers/414271/.

Columbia University. (2021, March 25). *Do you really need a degree to be a software engineer?* Columbia Engineering Boot Camps. Retrieved November 25, 2021, from https://bootcamp.cvn.columbia.edu/blog/do-you-need-degree-to-be-software-engineer/.

Hambrusch, S. (2018, January 18). *NAS report investigates the growth of computer science undergraduate enrollments*. CRN. Retrieved November 25, 2021, from https://cra.org/crn/2017/11/nas-report-investigates-growth-computer-science-undergraduate-enrollments/.

Harman, M. (2012, June 5), "The role of Artificial Intelligence in Software Engineering," *2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*, 2012, pp. 1-6, doi: 10.1109/RAISE.2012.6227961.

Sobania, D., Briesch, M., & Rothlauf, F. (2021, November 15). *Choose your programming copilot: A comparison of the program synthesis performance of GitHub* *copilot and Genetic Programming*. arXiv.org. Retrieved November 29, 2021, from https://arxiv.org/abs/2111.07875.

Piattini Mario, Peterssen Guido, & Pérez-Castillo Ricardo. (2020). Quantum Computing: A New Software Engineering Golden Age. Software Engineering Notes, 45(3), 12–14. https://doi.org/10.1145/3402127.3402131

Jones, C. (2014). The technical and Social History of Software Engineering. Addison-Wesley.

Leveson, N. G. (2017). The therac-25: 30 years later. Computer, 50(11), 8–11. https://doi.org/10.1109/mc.2017.4041349