# The Changing Role of the Software Engineer

Edward Meade[1], Emma O'Keeffe[1], Niall Lyons[1], Dean Lynch[1], Murat Yilmaz[2], Ulas Gulec[3], Rory V. O'Connor[1,4] [0000-0001-9253-0313], Paul M. Clarke[1,4] [0000-0002-4487-627X]

[1] School of Computing, Dublin City University, Glasnevin, Dublin, Ireland
{edward.meade5,emma.okeeffe7,niall.lyons4,dean.lynch49}@mail.dcu.ie

[2] Department of Computer Engineering, Cankaya University, Turkey
myilmaz@cankaya.edu.tr

[3] Hasan Kalyoncu University, Turkey
ulas.gulec@hku.edu.tr

[4] Lero, the Irish Software Engineering Research Center
{rory.oconnor,paul.m.clarke}@dcu.ie

**Abstract.** In this paper we will discuss the changing role of a software engineer. We will examine this from four major standpoints, the software development lifecycle, the influence of open source software, testing and deployment and the emergence of new technologies. We will first analyze what the role of a software engineer was in the past. We will examine limitations associated with software development life cycle models, and software failures that catalyzed increased importance for quality assurance. We then outline the current role of a software engineer. We discuss the impact of agile software development and automation on the software development cycle, the influence of open source software and how new technologies such as Function-as-a-Service and machine learning may impacted the role. Based on our research, we analyze why the software engineer role has changed and postulate prospective changes to the role of software engineer, and in particular how new responsibilities may affect the day to day work of future software engineers. We ultimately find that the role of a "software engineer" is nowadays widely varied and very broad, and it only generally indicates the type of work that the software engineer may undertake.

**Keywords:** Software Engineer, Agile, Open Source Software, Continuous Software Engineering.

## 1 Introduction

Software Engineering is defined by the IEEE as "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" [1]. Thus, the role of a software engineer could be defined as a person or team who apply systematic, disciplined, and quantifiable approaches to the development, operation, and maintenance of software. When analysing a software engineers' role, how they approach developing software must be discussed, as well as investigating the external technological factors influenced by the improvement of such technology.

When researching this topic, a multivocal approach was adopted, meaning that both peer reviewed and non-peer-reviewed artefacts were included. This gives a wide range of perspectives and was considered important especially as much of the practice community may not be heavily engaged with the peer-review/academic community. Examining the role of a software engineer is a broad undertaking that ultimately required the identification of specific search terms. An initial review of the literature found that there were relatively few published works dealing directly with the topic "the changing role of the software engineer". Therefore, specific search strings were employed so as to permit an examination of what were considered by the researchers to be among the more significant perspectives on the software engineering role over time. Inevitably, this selection decision introduces a limitation in this work which is discussed in further detail in the Section 4 Research Limitations.

To evaluate the future of a software engineer's role, we suggest that the past must be considered. Specifically, the influence of former software failures in changing the role of a software engineer. Along with this, in more recent years the introduction of free open source software has increased the need for flexibility in a software engineer's role. With changes to practice, new techniques can incur push back, as well as teething problems, especially if introduced at a rapid pace, and developers must be open minded to minimize these issues [2].

The role has also not only changed but branched out to more specific roles such as software tester. Testing is very much still a part of a software engineer's role; however, it has become its own distinguished role that requires experience in automation and vast testing knowledge. In the early years the role was not highly acclaimed but recently due to software failures, the desire for well-educated software testers has increased [3]. From identifying this separation of roles in the past, predictions can be proposed in relation to future role specialisation.

Automation will be discussed not only from a testing viewpoint but how increasing automation in the software development process has an impact on a software engineer's daily tasks as well as the delivery time of projects. With the introduction of Platform-as-a-Service and Function-as-a-Service in software deployment in recent years, software engineers have had to acquire knowledge to support these services.

The title software engineer is a broad term which can encompass a vast variety of responsibilities that depends on many factors including companies own definition, technology trends and the progression of development methodologies. The role has advanced and evolved over the years and will continue as technology continues to change. Due to the frequent changing in tools, methods and technology the constant factor of a software engineers' role is to be flexible and open to consistent learning. Section 2 outlines the literature review methodology, Section 3 presents an analysis of the selected literature, while Section 4 identifies acknowledged research limitations. Sections 5 and 6 present future work and conclusions.

## 2 Literature Review Methodology

The basis of this research was conducted in the context of a final year B.Sc. in Computer Applications assignment in Dublin City University, Ireland. The assignment involved four team members who were allocated a research paper title: “The Changing Role of the Software Engineer”. This topic was considered relevant to their studies as many would soon become professional software engineers and an examination of the role and its possible future direction is of central importance not just to their studies, but also to their impending careers.

Structuring the literature review involved breaking down the overall task into several smaller steps so as to enable the team to explore the literature to systematically extract relevant information. Firstly, key search strings were utilised: ‘Changing Role of Software Engineer’, ‘Future of software engineer’, ‘software industry change’, ‘Automation in software engineering’ and ‘Evolution of Software Engineer’ so as to identify a baseline set of find a key set of aretfacts (i.e. research papers and other non peer-reviewed sources). Google, Google Scholar and digital libraries of publications including, ScienceDirect, ACM, IEEE and Wiley were used to find these publications. A short list of forty works was derived for each of the search strings employed. This shortlisting was achieved by jointly evaluating the relevance of each work and its citation count (where available).

Following the completion of the initial phase of literature review, the shortlists for each search term were further evaluated. ISI Software Engineering journal impact was also used as a broad filtering mechanism: high impact factor journals were considered more important as sources. The relevance of publications was examined by reading each abstract and conclusion. Each publication was sorted due to the number of citations it had and the year it was published. The next step of the process involved reading the publications in detail and making further evaluations in relation to relevance. All publications deemed to have substantial relevance to the topic were added to a set, with the number of citations used as a proxy to determine the overall popularity of the work. Some limitations in this approach are acknowledged in Section 4 of this paper, but overall the result of this systematic exercise was a listing of 30 sources that the research team set to work on reviewing in detail for key information relevant to producing a research paper that broadly examined the role of the software engineering, how it has changed to date, how it is performed at present, and how it may evolve into the future. The results of the analysis of this material are present in Section 3.

## 3 Analysis

### 3.1 Role of a Software Engineer in the Past

Evaluating the role of a software engineer means looking to the past and identifying what influences the job specification. As technology began to emerge, the technology industry concentrated mainly on meeting institutional needs, which led to research in development methodologies to produce planned and scheduled software [4]. Software development tends to be a group activity with communication being a vital part of a software engineers’ role. However, before the implementation of methodologies projects often suffered due to lack of communication between other developers and users [5].

The evolution of software development life cycle models has directly impacted the software engineer's role. Though it is worth reflecting on the fact that principles formalised in a lifecycle model or methodology might predate the formal arrival of the model [6], for example, agile software development concepts such as test driven development existed prior to the introduction of agile methods. Further into the 1990's, software engineers saw the introduction of documentation as a standard in their role to ensure quality assurance [7]. This was at least partly because of software failures such as Therac-25 which led to not only serious injury to humans but also deaths [7]. Therac-25 was also an example of why software engineers working as individuals and not as a team can lead to adverse outcomes for users. Testing may have come to be perceived as more important during this era, but this might be difficult to definitely prove in retrospect as the volumes of software being produced were also increasing and therefore, the need for larger scale software system may have warranted more structured development processes. Whatever the case, software engineers were might expect in certain cases to work in large teams with implications for their roles and responsibilities.

Although it was becoming normal for organisations to employ software testing professionals, this did mean that software engineers were no longer required to test, especially in smaller companies [8]. With the role of software tester perceived as "consolation prizes for those not considered good enough to hire as software engineers", the idea that the job is not highly respected and that there may be no career growth often deters new graduates who may believe it to be lesser job [3]. This could in turn cause the misconception that a software engineer never has to do any testing as it is below them. Indeed, prior to the introduction of unit testing frameworks, it may have been the case in some settings those writing software code were mostly removed from the testing concern.

In 2000, Microsoft declared that "the role of a software engineer … deals with file transfer and management, troubleshooting/bug fixing, improving methods and consultancy" [9]. In a 2011 article, Microsoft employees discuss code ownership within the company, this time with software engineers being fully responsible for their own code, including testing [10]. It seems therefore that as time progressed, software engineers may have ultimately had to become more involved in quality assurance for their own work products.

In a 2008 article, (sometimes) free open source software was growing in popularity, thus increasing the need for software engineers to understand and integrate software not developed within their own company [11]. This is an example of a demand on software engineers to continuously keep up to date on emerging concepts and facilities, something we suggest from our literature review to be a regular requirement on software engineers.

### 3.2 Current Role of a Software Engineer

As software development has evolved, it seems from our research that the role of a software engineer has become broader and more heterogeneous in nature. There are now many aspects to software engineering. Even as early as 2002, a software engineer could include being a developer, a tester, a maintenance analyst, a reviewer and a designer [12], and indeed was sometimes referred to as a software development engineer. In the present time, software engineers can no longer solely rely on their programming expertise alone. They must also have a keen understanding of the strategic impact of technology decisions on the business, while building a relationship with business peers and understanding the real-life application of what they are doing [13]. A focus on the efficacy of real-world application of their code may not have been so continually prevalent in the classical software engineer's remit. We have also witnessed significant change in the delivery of software, even in the utilisation of open source software. The altering of open source software has provided the flexibility to software engineers to no longer wait until a project is feature complete to ship; improvements are made incrementally and driven internally and externally as feedback is gathered [14].

While automation of individual programming tasks (compilation, testing, documentation, etc.) has been part and parcel of software engineering for many years, we are currently witnessing an automation intensification across many aspects of the software development process [15]. Automation permits software feature delivery at rates, which just a few short years ago may have been unachievable when constrained by manual intervention. In a recent study it was seen that a software engineer increases their productivity, while simplifying the deployment stages through an automated pipeline, with the percentage of software engineering teams that do not automate deployment shrinking from 26% in 2016 to 11% in 2017 [16]. Through the use of automation, software engineers use continuous integration, a development practice where engineers integrate code into a shared repository many times a day. Each integration is created through automated building scripts and verified with automated tests, delivering the benefit of more rapid error detection and improved error source identification [17]. Continuous integration, coupled with advances in version control, has improved the capacity of software engineers to compare different versions of the code, to find errors and to minimise downtime [18].

Agile based development is by design capable of reducing communication related delays and increasing discussion between software engineers, but it can also reduce the amount of documentation produced [19]. When working in geographically distributed teams, this may introduce challenges in communication since some remote communication, we suggest, may in fact benefit from documented artefacts. We therefore see that the evolution towards agile software development may have introduced new challenges for software engineers working on distributed teams.

Platform-as-a-Service (PaaS) and the still emerging Function-as-a-Service (FaaS) are third party computing services that software engineers are using for deployments [20]. With both of these advances, software can run on hardware that is largely or fully managed by third party providers but to do so, developers may need to acquire new skills such as microservices based systems development [40]. Such development paradigms are largely classifiable as distributed systems and may require that the software engineer to take on new design principles which they have not previously experienced.

As technology advances the roles of software engineers and the needs of companies are changing. Google have introduced a new software engineering role called site reliability engineering [21], building engineering teams to develop tools and systems reducing toil and repetitive work, while monitoring large distributed systems. The building of resilient architectures and thinking about scalability from the very beginning are key aspects to the role. As referred to above, the automation of as much as possible such as deployments, maintenances, tests and scaling is also core to this role [21].

Large distributed software systems must be designed and built for change, scalability and efficiency. A niche area of artificial intelligence (AI) and machine learning (ML) engineering practices are evolving a traditional software engineers' role. Large software systems benefit from this maturing software discipline as in time, the thoughts of shifting the burden of evolution from the software engineers to the systems themselves and exploring what it would mean to build systems that can take responsibility for their own evolution [22]. As this area of software engineering is growing, the influences and challenges of AI and ML in large systems will, we suggest, have a significant effect on the role of a software engineer in the future.

## 3.3 Why the Role of a Software Engineer Changed

Our research suggests that the transition from traditional lifecycle models to agile software development is one of the major reasons for evolution in the software engineer's role. A waterfall model is a sequential process, where each phase of the process relies on the previous phases being completed. All requirements are believed to be well defined at project outset, and largely unchanging thereafter [23]. An agile model can be described as "an iterative and incremental (evolutionary) approach to software development which is performed in a highly collaborative manner by self-organizing teams within an effective governance framework with "just enough" ceremony that produces high quality software in a cost effective and timely manner which meets the changing needs of its stakeholders" [24]. Agile assumes there will be change in the requirements and is an iterative process. With agile software development software engineers no longer have a clear set of requirements at the outset (indeed a key component is that regular feedback will refine requirements to meet end users' needs). This, we find, has profoundly affected the role of a software engineer who is now required to deal with constant feedback from clients (or their designate) and moreover, to engage daily with team members in formalised ceremonies.

We also find that explicit new responsibilities have been created by agile software development within software engineering teams, where there are many different roles, especially on self-organizing teams. The roles that may arise include mentor, co-ordinator, translator, champion, promoter, and terminator [25]. The introduction of these roles means that a software engineer no longer only must develop code but also must perform their role as a part of an agile team. It may also have implications for accountability for software engineers, who now face the pressure from their peers as well as their managers. We suggest that these changes to the social structure in agile development teams can introduce new pressures for individual developers.

Open source software has also has a major impact on the role of a software engineer. Open source software began to emerge as a viable option for companies as early as 2006 [26]. Its emergence has created a vast community, with lots of collaboration and contribution to many projects. Companies use these communities to find new talent while some companies release their own open source software so that it can be utilised by many other companies and increase revenue opportunities for themselves [27], and with impacts for software engineers. As companies use more open source software, a software engineer must develop a wide range of technical skills to deal with this. They must ensure that the open source software is seamlessly knitted together with their own project which is not necessarily are trivial undertaking. This change means that software engineers must constantly look for areas where they can expand their own knowledge and consistently look to grow. Furthermore, working on open source projects can indirectly benefit software engineers as companies may try to recruit them from their portfolio of work in the open source community.

The software industry is striving to make products cheaper, better and quicker. One way this has been achieved has been through "the automation of control of systems, consisting of hardware and a growing software part" [28]. One of the major impacts – though by no means a recent development - has been the use of version control. For some time, most software engineers have a single common code repository [29]. Version control allows software engineers to easily track changes and who made the changes. It enables software engineers to have access to the most up to date version of the code repository. We suggest that this has led to software engineers being required to collaborate more, perhaps driven by the rise in the scale and complexity of software projects over time. With continuous software engineering [39], software engineers have had to change even more, now required to continually commit their code and run the risk of breaking other aspects of the greater build on a very frequent basis (this risk always existed, but was only occasionally manifested in traditional big bang approaches). Perhaps therefore, software engineers are now held more (immediately) responsible for the code they write. Related to this are the automated building and testing processes, which allow a software engineer to quickly identify if their code has affected any other parts of the project [30]. Due to this, software engineers may seek to produce better quality of code in the first instance.

Our research has also highlighted the impact of DevOps on the role of the software engineer. According to Amazon, DevOps is "is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes". [30] With DevOps, there is a demand to harmonise development and operations environments, and so software engineers may need to collaborate more closely operations engineers to establish and sustain common environments. The general trend towards faster paces of feature deliver is represented in Figure 1.

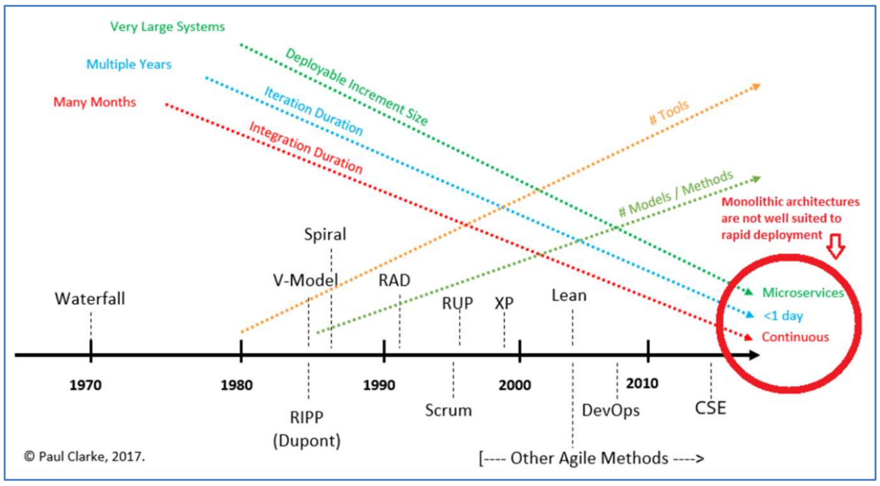

**Figure 1**. Historical SE trends (integration and deployment frequency)

A further major reason for change to the software engineer role relates to the emergence of new technologies such as FaaS and Machine Learning (as previously noted). Software engineers may not have been trained in distributed systems development or in Machine Learning or AI. These emerging demands on the software engineer are not insignificant and represent a fundamental change to the day to day work that a software engineer might be required to undertake.

## 4 Research Limitations

This research was primarily conducted by a team of four final year undergraduate computing degree students. This introduces a risk that the students were not equipped with the desirable levels of research expertise. To offset this risk, a number of steps were implemented. Firstly, the students were provided with a tutorial session on conducting literature reviews and undertaking academic research. Secondly, the students operated under the direction of their supervisor with whom they could avail of weekly research engagements over the 6 weeks of the assignment duration.

Despite the time bounds imposed on the work (owing to the fact that it was conducted as an undergraduate assignment), the methodology employed is nevertheless considered to be rigorous and repeatable. The search strings and databases employed are clearly identified (ref. Section 2). Furthermore, papers identified were systematically sorted using a combination of citation count, journal impact factor and perceived relevance to the topic. Using citation count and impact factor as discriminating criteria is also a limitation as it is not necessarily the case that high citation counts / impact factors guarantee better quality publications or higher relevance. However, when used in tandem with perceived relevance (as judged by the research team), a robust paper selection mechanism was designed and employed.

A further limitation arises in the limits placed on the total number of papers to be examined for each of the search strings. During this research, a limit of the top forty results for each search string per database was applied. Additional relevant papers may have been identified if a larger threshold had been applied, but having 40 papers per search string per database is considered sufficient to form a robust view of the topic. The further limitation to 30 sources for consideration in the ultimate core analysis effort also limits the scope of research but it does not diminish the research credibility and given the contents of this paper, we suggest that it has not inhibited the research from producing interesting and valuable findings. However, given the constraints applied, the work present herein - although academically robust - cannot be considered exhaustive and no such claim is therefore made in this paper. Finally, the search strings themselves also necessarily limit the scope of the work. These were largely decided upon in consultation with the supervisor, were necessary to allow the research to advance towards meaningful findings given other constraints, and are considered by the broader team to be areas of direct and important relevance to the overall research theme.

## 5 Future Work / Directions for Future Research

In the future we would like to expand the number of search strings used, as well as the limitations set during the research process. With more time, and experience, the core topic could be more deeply evaluated and further important perspectives would be identified.

We would have liked to contrast the changes that have occurred inside the role of the software engineer with the changes that have occurred across similar professions. This could potentially have given a clearer picture as to why these changes happened. They may indicate if the root of their cause was associated with problems closely related to the software engineering role, or if they stemmed from a more ubiquitous source in the wider engineering industry.

Due to the constantly changing and expanding role of a software engineer, and its necessity to adapt to the very latest technologies, it's important to watch the current trends meticulously. In this respect, our research offers only limited coverage for recently published papers on trending technologies. This made it difficult to draw any conclusions as to which trends will be sustained and consequently, in the future, we would like to revisit the current and emerging trends and analyse how they might influence the future of software engineering. Indeed, the examination and prediction of future trends might itself be a substantively different research focus.

We would also like to revisit our earlier research investigating personality types and role suitability in software engineering [31], this time in the context of the changes that we have identified in the role of software engineer.

## 6 Conclusions

Software development is a complex socio-technical activity [32], and in earlier work we have examined the complex interaction between a software development process and its situational context [33-35]. Part of the challenge of organising the software development process involves deciding upon the specific roles involved in the process and the responsibilities of individuals fulfilling these roles. Therefore, just as the process itself is subject to regular change [36-38], so too are the roles subject to change.

We have therefore examined the role of the software engineer and how this has changed over time, including how it might change into the future.

In the early years of software development, the software engineer role may have been no bigger than one individual working in isolation to write small amounts of program code to execute on dedicated hardware. However, as the volumes and importance of software increased over time, ever larger teams of software engineers were required to work together, meaning that communication and cooperation became more vital considerations. This led to a refinement of methodologies such as the waterfall, and later agile, which directly affected the early evolution of the software engineering role. For a time and in certain settings, the software engineer may have been limited solely to writing program code for requirements presented to them and with their code quality being evaluated by a separate software testing function. However, at some point in the introduction of agile software development, there would appear to have been a shift towards more explicit programmer responsibility for their own code.

Our research would suggest that at the present time, the role of a software engineer is widely varied and that perhaps it suffers from an absence of common agreement. A software engineer can now be defined as a developer, tester, maintenance analyst, reviewer and a designer [13]. One of the most significant changes seen at present is the rate at which software is shipped. Software is being continually shipped, instead of waiting for the project to be feature complete [14]. Automation of certain programming tasks and a shift to agile development and cloud based infrastructure have helped make this possible. With these changes, the architecture behind the development has also been reimagined, giving rise to Platform-as-a-Service and Function-as-a-Service offerings. Artificial intelligence and machine learning advancements may also assist with the scalability and efficiency of these systems. However, these technologies are still in their early stages, and software engineers continue to examine how to use these technologies to their full potential.

Our research indicates that the transition from traditional development approaches towards agile is one of the main reasons why the classical role of a software engineer is to some extent antiquated. Software engineers have now become more adaptive and have more responsibilities in the context of the broader project. One example of this is the growing expectation to undertake some level of code unit testing. Open source software has also played a part in the changes over the last decade, creating a vast community of collaborators and contributors to a wide range of projects, with many companies evaluating open source contributions for the purpose of recruitment as well as opportunity their own growth.

The widespread introduction of automation, not least in version control and continuous integration, has allowed greater collaboration among engineers, but also requires that engineers have a greater understanding and responsibility of the broader code base with which they are working. Automation of build processes has allowed engineers to spend less time worrying about compatibility and more time working on the problem at hand. Automation in testing at various levels from individual units, to integration to front end, as exemplified in DevOps has also resulted in a quicker development cycle [30].

New technologies are always emerging, such as artificial intelligence and machine learning, and software engineers are expected to continually keep up to date. This may entail - in some circumstances - learning about these new technologies outside of their regular working hours. And while this situation might not be desirable, it is an exciting time to work as a software engineer, as some of these technologies have a lot of potential to change the way work is done on a day to day basis. Looking to the medium to long term, predicting the future role of a software engineer is not an easy task, principally because software development continues to exhibit strong levels of innovation. However, certain trends would appear set to be sustained into the future. Continuous software engineering techniques [40] will shrink the development cycle ever further. Perhaps in the longer term, AI itself will construct software systems (or parts of systems) through the assembly of components of known functionality and quality. Any such advancement might herald the demise of the software engineer, but we assert that there is some considerable distance to cover before any such future might emerge.

**Acknowledgement.** This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 to Lero - the Irish Software Research Centre (www.lero.ie).

## References

1. IEEE.: IEEE Standard Glossary of Software Engineering Terminology. pp. 70–70, (1990)
2. Fenton, N., Pfleeger, S., Glass, R. Science and substance: A challenge to software engineers. IEEE Software,11(4), 86-95 (1994).
3. Glass, R., Collard, R., Bertolino, A., Bach, J., & Kaner, C.: Software testing and industry needs. IEEE Software,23(4), 55-57 (2006).
4. Kling R. Computerization and controversy: value conflicts and social choices. Academic Press, San Diego (2011).
5. Basili V, Musa J.: The future generation of software: a management perspective. Computer 24:90–96. doi: 10.1109/2.84903. (1991).
6. Clarke, P., O'Connor, R.V., Yilmaz, M.: In Search of the Origins and Enduring Impact of Agile Software Development. ACM proceedings of the International Conference of Software and System Processes (ICSSP 2018), Gothenburg, Sweden. 26-27 May 2018. pp.142-146.
7. Leveson N, Turner C.: An investigation of the Therac-25 accidents. Computer 26:18–41. doi: 10.1109/mc.1993.274940 (1993).
8. Rooksby, J., Rouncefield, M., & Sommerville, I.: Testing in the Wild: The Social and Organisational Dimensions of Real World Practice. Computer Supported Cooperative Work (CSCW),18(5-6), 559-580 (2009).
9. Orsted M.: Software Development Engineer in Microsoft. In: Proceedings of the 2000 International Conference on Software Engineering, 2000. IEEE / Institute of Electrical and Electronics Engineers Incorporated, pp 539–540. Limerick (2000).
10. Bird, C., Nagappan, N., Murphy, B., Gall, H., & Devanbu, P.: Dont touch my code! Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering - SIGSOFT/FSE 11. doi:10.1145/2025113.2025119 (2011).
11. Ebert, C.: Open Source Software in Industry. IEEE Software, vol. 25, no. 3, pp. 52–53., doi:10.1109/ms.2008.67 (2008).
12. Orsted, M.: Software Development Engineer in Microsoft. A Subjective View of Soft Skills Required. 6 Aug., doi:10.1145/337180.337445 (2002).
13. McColgan, C.: Business Savvy Developers to Be in High Demand in 2017. Recruitment-buzz.co.uk, 4 Jan. 2017, recruitmentbuzz.co.uk/business-savvy-developers-high-demand-2017/.
14. Roumeliotis, R.: Software Engineers Must Continuously Learn and Integrate. O'Reilly, www.oreilly.com/ideas/software-engineer-developer-coding-architecture-mobile-open-source.

15. Carrillo De Gea, J. M., Nicolás, J., Fernández-Alemán, J. L., & Toval, A.: Automated support for reuse-based requirements engineering in global software engineering. Journal of Software: Evolution and Process,29(8). doi:10.1002/smr.1873 (2002).
16. Visser, J.: 5 Automation Trends in Software Development, Quantified. O'Reilly, www.oreilly.com/ideas/5-automation-trends-in-software-development-quantified.
17. Continuous Integration Essentials. Https://Codeship.com, codeship.com/continuous-integration-essentials.
18. Atlassian. (n.d.). What is version control | Atlassian Git Tutorial. Retrieved from http://www.atlassian.com/git/tutorials/what-is-version-control.
19. Rizvi, B., Bagheri, E., & Gasevic, D.: A systematic review of distributed Agile software engineering. Journal of Software: Evolution and Process,27(10), 723-762. doi:10.1002/smr.1718 (2015).
20. Ghosh, P.: Serverless Computing and Serverless Architecture: An Overview of BaaS, FaaS, and PaaS. www.dataversity.net, 17 Jan. 2018, www.dataversity.net/serverless-computing-serverless-architecture-overview/#.
21. Beyer, B., Murphy, N. R., Rensin, D. K., Kawahara, K., & Thorne, S.: The site reliability workbook: Practical ways to implement SRE. Sebastopol, CA: OReilly Media (2018).
22. Zhang, Du, and J.J.P Tsai.: Machine Learning and Software Engineering. IEEE, doi:10.1109/TAI.2002.1180784 (2002).
23. Palmquist, S., Lapham, M. A., Garcia-Miller, S., Chick, T. A., Ozkaya, I.: Parallel Worlds: Agile and Waterfall Differences and Similarities. Software Engineering Institute, Oct. 2013, pp. 1–15.
24. Lapham, M. A., Bandor, M., & Wrubel, E.: Agile Methods and Request for Change (RFC): Observations from DoD Acquisition Programs. doi:10.21236/ada609878 (2014).
25. Hoda, R., Noble, J., & Marshall, S.: Self-Organizing Roles on Agile Software Development Teams. IEEE Transactions on Software Engineering,39(3), 422-444. doi:10.1109/tse.2012.30 (2013).
26. Fitzgerald, B.: The Transformation of Open Source Software. MIS Quarterly, vol. 30, no. 3, p. 587., doi:10.2307/25148740. (2006).
27. Bradford, L.: How Open-Source Development Is Democratizing The Tech Industry. Forbes, Forbes Magazine, 27 Mar. 2018, www.forbes.com/sites/laurencebradford/2018/03/26/how-open-source-development-is-democratizing-the-tech-industry/#f3ce1fd3bb6a.
28. Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M., & Göhner, P.: Challenges for Software Engineering in Automation. Journal of Software Engineering and Applications,07(05), 440-451 (2014).
29. Łukiański, M.: Why Do We Automate Software Development? . Droptica Blog, Droptica, 24 Jan. 2019, www.droptica.com/blog/why-do-we-automate-software-development/.
30. What Is DevOps? - Amazon Web Services (AWS). Amazon, Amazon, aws.amazon.com/devops/what-is-devops/.
31. Yilmaz, M., O'Connor, R.V., Clarke, P.: An exploration of individual personality types in software development. In: proceedings of the 21st European Conference on Systems, Software and Services Process Improvement (EuroSPI 2014), 25-27 June 2014, Luxembourg (2014).
32. Clarke, P., O'Connor, R.V., Leavy, B.: A Complexity Theory viewpoint on the Software Development Process and Situational Context. In: proceedings of the International Conference on Software and Systems Process (ICSSP), Co-Located with the International Conference on Software Engineering (ICSE), pp. 86-90, DOI:10.1145/2904354.2904369 (2016).
33. Clarke, P., O'Connor, R.V.: The situational factors that affect the software development process: Towards a comprehensive reference framework, Information and Software Technology, Vol. 54(5), May 2012, pp.433-447.
34. Giray, G., Yilmaz, M., O'Connor, R.V., Clarke, P.: The Impact of Situational Context on Software Process: A Case Study of a Small-sized Company in the Online Advertising

Domain. Accepted for publication in the proceedings of the 25th European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2018).

35. Marks, G., O'Connor, R.V., Clarke, P.: The impact of situational context on the software development process - A case study of a highly innovative start-up organization. In: Proceedings of the 17th International SPICE Conference (SPICE 2017), pp.455-466; 4-5 October 2017, Palma de Mallorca, Spain.
36. Clarke, P., O'Connor, R.V., Leavy, B., Yilmaz, M.: Exploring the Relationship between Software Process Adaptive Capability and Organisational Performance. IEEE Transactions on Software Engineering, 41(12), pp.1169-1183, doi: 10.1109/TSE.2015.2467388 (2015)
37. Clarke, P., O'Connor, R.V.: Changing situational contexts present a constant challenge to software developers. In: Proceedings of the 22nd European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2015), CCIS (Vol. 543), pp. 100-111, 30 September - 02 October 2015, Ankara, Turkey. (2015)
38. Clarke, P., O'Connor, R.V., Solan, D., Elger, P., Yilmaz, M., Ennis, A., Gerrity, M., McGrath, S., Treanor, R.: Exploring Software Process Variation Arising from Differences in Situational Context. In: Proceedings of the 24th European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2017), pp.29-42, 5-8 September 2017, Ostrava, Czech Republic.
39. Clarke, P., Elger, P., O'Connor, R.V.: Technology-Enabled Continuous Software Development. In: Proceedings of the International Conference on Software Engineering (ICSE) Workshop on Continuous Software Evolution and Delivery (CSED) (2016)
40. O'Connor, R.V., Elger, P., Clarke, P.: Continuous Software Engineering - A Microservices Architecture Perspective. Journal of Software: Evolution and Process, 29(11), 2017, pp.1-12.