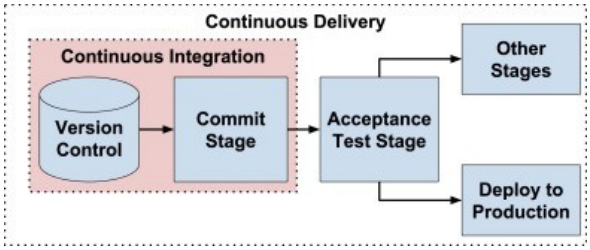
Simplifying DevOps: How NLP and AI can support Development and Operations

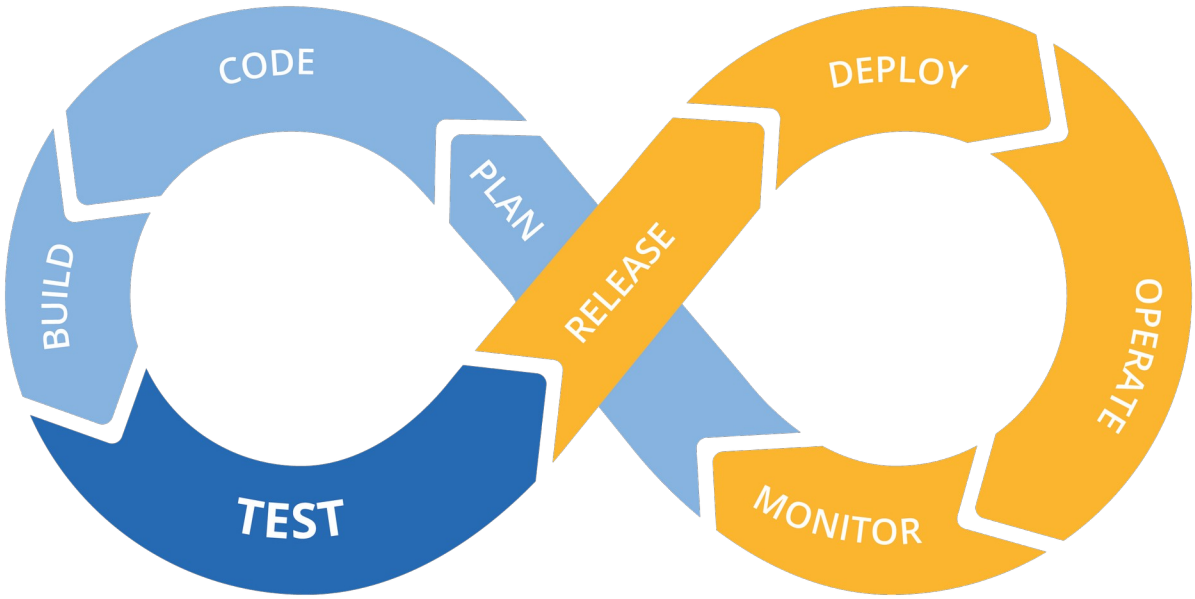
DevOps

DevOps is a modern approach to software development focused on Continuous Integration, Continuous Delivery (CI/CD) supported by automation, and close collaboration between Developers and Operations. CI/CD consists of frequent integration of new code and releases aiming to get early feedback, improve productivity, and stakeholders' satisfaction.



Although useful, DevOps is far from being perfect. While technical problems are common to other development strategies, new "human problems" are a symptom of new concepts that teams are unfit to support. Organizations, stakeholders, as well as IT, may block the adoption of DevOps (Laukkanen et al., 2017). Developers are concerned that the new practices would increase pressure and subtract time from development, and both Developers and Operations lack interest in each others' competencies complicating the creation of cross-functional teams (Smeds et al., 2015; Hemon et al., 2020a). Dissatisfaction, errors, and low performances are the result of a poorly received adoption of DevOps (Hemon et al., 2020b; Taribuka et al., 2020). Legacy software also impedes its adoption because it cannot be adapted to CI/CD without considerable investment (Kuusinen et al., 2018). Creating cross-

functional teams increasing the size of teams also correlates with lower productivity (Rodriguez et al., 2012).



DevOps phases

Requirement collection in Planning phase

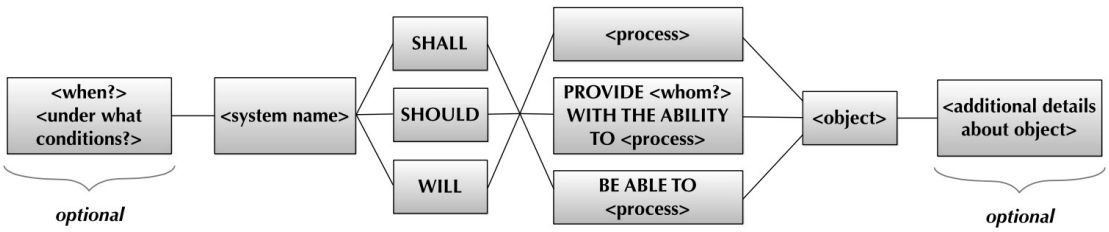
Standish estimates that only 28% of projects are successful (Standish Group, 2001, as cited in Lobo & Arthur, 2005) with enormous economical impact (Stadish Group, 1995, as cited in Kasser & Williams, 1998). “Poor requirements” is the primary risk according to Lobo & Arthur (2005) confirming the findings of Standish, and it is also the only technical in a list otherwise only composed by project management, economic, and governance risks. Lindquist attributes to poor requirements 71% of failures (2005, as in Muqueem 2014) while Davis raises the percentage to 90% (2006, as in Muqueem 2014)

As found by Mich et al. (2004), only 5% of requirements are written in formalized language while 95% is Natural Language (NL). 31% of respondents to Dawood & Sahraoui’s survey (2017) do not use a systematic approach. Zhao et al. (2020) define NL as the “lingua franca” for requirements excluding a change for the future

except for the adoption of templates. Requirements expressed in NL are 61% according to Kassab et al. (2014), who also finds that the most common approach to refine them is “brainstorming” (65%) with only a small percentage (33%) using semi-formal notations such as UML.

NL processing (NLP) could offer support in this phase. As observed by Mich (1996), one challenge of NLP is to converge grammatical representations of requirements into the same knowledge model, a task requiring an NLP able to process a wide variety of characteristics of NL. Mich postulates that the system must cover requirement elicitation and validation, text and graphic representations, and Object-Oriented methodologies. The spirit of these characteristics is to give maximum support to the analyst through common NL without requiring new skills. Mich also highlights the challenges that can be summarized in “long sentences”, “detection of ambiguity and inconsistency”, and “determination of entities”.

In the field of NLP for requirement engineering (NLP4RE), Arora et al. (2015) suggest that the usage of templates helps reduce ambiguity and they propose Rupp’s template. Templates constrain the sentence’s structure to a form that can be easily parsed by NLP to validate the consistency of the template itself with simple text chunking. Denger et al. (2005) agree that templates and standards contribute to better completeness, understandability, modifiability, and verifiability.



Rupp’s template

Femmer et al. (2017) propose the concept of Requirement Smell (RS) inspired by the concept of Code Smell (Fowler et al., 1999). An RS is a sign of quality violation and can pose a problem for verification even without implying a defect in the final product. Femmer et al. propose a framework to automatically detect RS and reduce the cost of requirement reviews. The solution is based on part of speech (POS) tagging, morphological analysis, usage of dictionaries, and lemmatization to normalize and parse NL. Results are encouraging and span from 96% precision detecting subjective language to 50% for comparatives and superlatives.

The study of Hamza & Mustafa (2019) focuses on the automatic transformation of requirements into UML diagrams. The process makes usage of POS, chunking, grammar rules, and stemming to prepare the text for the transformation. The average overall precision is 72% with coverage of requirements of almost 70%.

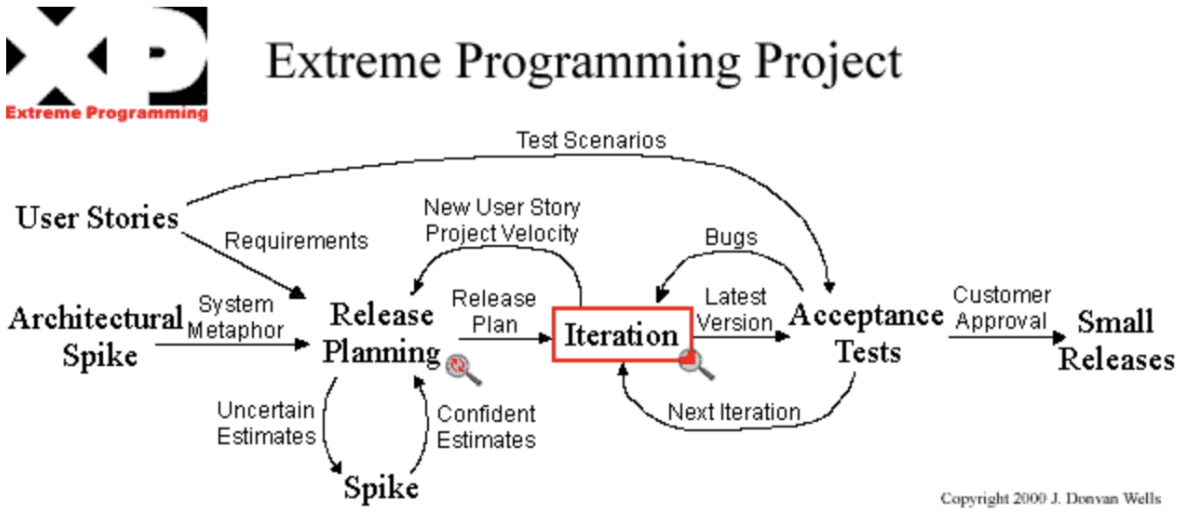
As observed by Ferrari (2018) NLP is living a “golden age” with a proliferation of support in Java and Python and the creation of tools capable of enriching the text with a semantic layer usable to spot similarities between requirements and to improve traceability. Zhao et al. (2020) observe that the interest for NLP4RE increased over time from two publications a year until 2004 to 24 a year in 2019, and the development of 130 tools. However, for now, there is no significant adoption in the industry even if there seems to be a market need as observed by Mich et al. (2004).

Improving the Code, Build, and Test phases

There are several techniques to accelerate coding. Scaffolding in Ruby and Maven Archetypes for Java are examples to generate the application’s skeleton. Modern frameworks like Spring Boot come with a predefined structure to let the developer

focus on the core functionality. Code generation from UML is widely used to generate algorithmic parts, while schemas like XSD, WSDL, or GraphQL, are the base to generate models or API. Shinde & Sun (2016) suggest expanding the concept with code generation based on templates to handle recurring problems like input parsing and database IO. Templates avoid defects like datatype mismatch, missing error handling, and incorrect validation, boosting productivity, quality, and code reuse.

Denger & Olsson (2005) recommends performing Quality Assurance as early as possible to minimize cost and impact. They recommend prototyping as a way to validate requirements and get early feedback from stakeholders. Prototyping is what Agile practices call “spike” and it is recommended by eXtreme Programming, Scrum, and DevOps. (Wells, 2000; Schwaber & Sutherland, 2020; Kuusinen et al., 2018)



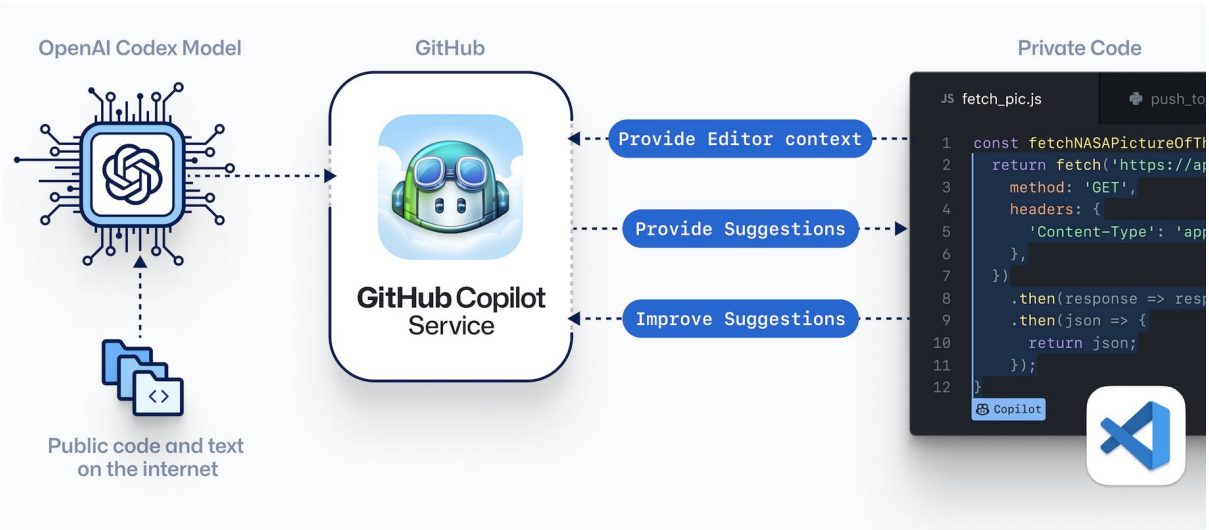
XP Model

72.9% of code vulnerabilities are in the implementation, with the rest equally divided between design and configuration (Kuhn et al., 2017). Tools like SonaQube (SonarSource, 2021) can perform statical analysis, find issues and suggest remediation. The field of Automated Program Repair (APR) aims to evolve the

approach detecting and solving bugs automatically. Although promising, APR can address only a small fraction of known issues (Benton et al., 2020).

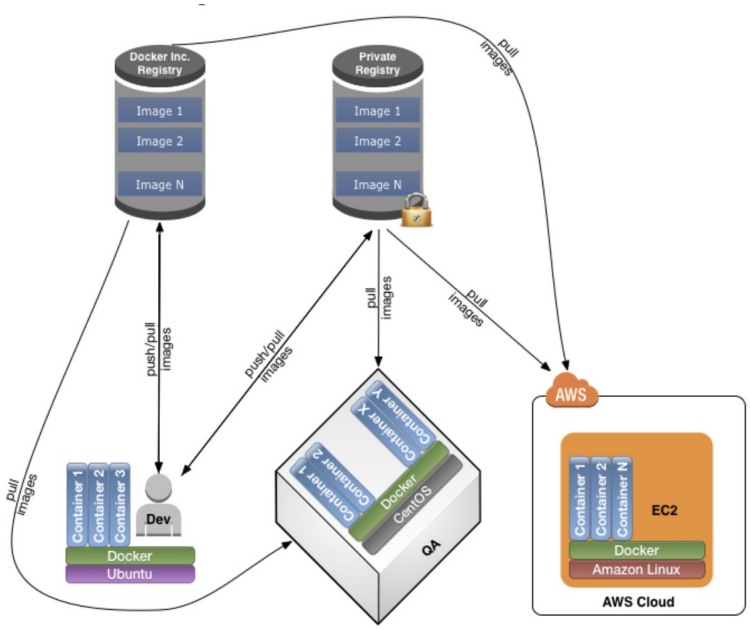
Dependency scanners are instead an established solution: OWASP tools supported by databases like NVD (OWASP, 2021; NIST, 2021) are precious to detect potential risks. However, in this area, there is still a need for manual work to assess if vulnerabilities are potential or effective (Elron, 2021).

Looking at the future, OpenAI's Codex is a deep learning AI trained with Gitlab's public repositories and able to generate code from NL. The initial version can translate into code "easy interview questions" (Chen et al., 2021). At its presentation, it was able to solve 37% of OpenAI's benchmark. Codex can correctly generate software and integrate third party's API. It was demonstrated how Codex can enhance MS Word to perform simple tasks defined orally in NL (OpenAI, 2021). A version of Codex powers Gitlab's Copilot, a tool integrated into the IDE offering advanced contextual auto-completion functionalities. Copilot can give valuable suggestions and implement entire portions of code (Github, 2021), but it is still in an early stage. An evaluation of the quality of the suggestions revealed that suggestions contain vulnerabilities in 40% of cases (Hammond, 2021).



Improving Release, Deploy, and Operate phases

Docker promotes the single responsibility principle to create complex systems composed of loosely coupled modules. Its main advantage for development is to offer a lightweight virtualization layer that eliminates differences between developers' machines and production with a level of isolation that avoids unexpected conflicts (Merkel, 2014).



Kubernetes is an orchestration platform for containers frequently used together with Docker to manage applications composed by microservices. Kubernetes supports the releases with automatic deployments, rollbacks, and rollovers. It offers automatic service discovery and load balancing, and it can monitor services, detect failures, and heal the disruption without human intervention (Shah & Dubaria, 2019). Current commercial solutions supporting Kubernetes offer no protection against application failure (Nabi et al., 2016), although, a combination of architecture based on microservices, Docker, and Kubernetes, is the only way to achieve High Availablity, (99.999% uptime) (Vayghan, 2018). In the future Kubernetes may be enhanced with

AI-Schedulers to get better results (Jorge-Martnez, 2021) just like AI can already be used to solve common networking problems in Software-Defined Networks (Latah & Toker, 2019).

Improving Monitoring phase

Log monitoring is necessary to detect anomalies. The current state of the art are tools like Splunk, able to index and correlate events (Splunk, 2021) but they are passive tools that need a human expert. The next evolution will be an AI-based tool capable of detecting anomalies with no or limited human intervention. Padman et al. (2019) concluded that even unsupervised AI can be effective to find events that may otherwise go unnoticed.

The study case

The study case (SC) is a large organization (5.000+ employees) in the transportation sector. SC develops software in support of the business.

In SC, requirements are collected in non-standardized NL with no template. Requirements are refined by analysts and developers together, and suffer clarity problems resulting in long lead time. Given the lack of any standard, even the simple adoption of templates would improve the quality of requirements reducing the ping-pong between groups.

The development processes are “in push” (as defined by Spearman & Zazanis, 1992). The characteristic of push processes is the risk of work pressure, but also the need for alignment and understanding of the downstream processes. The adoption of DevOps is helpful to improve collaboration, but it is successful only in a small

number of teams working on greenfield projects and it results in frustration elsewhere.

Code generation, prototyping, static code analysis, and dependency scanners are commonly used with good results confirming the finding of the research. It is evident, however, that teams struggle with technical debt. APR and automatic assessment of vulnerabilities in dependencies would be extremely valuable if developed. Codex and Copilot could also be helpful, but the impossibility to evaluate their cybersecurity risk is a reason to block their adoption in traditional contexts such as SC.

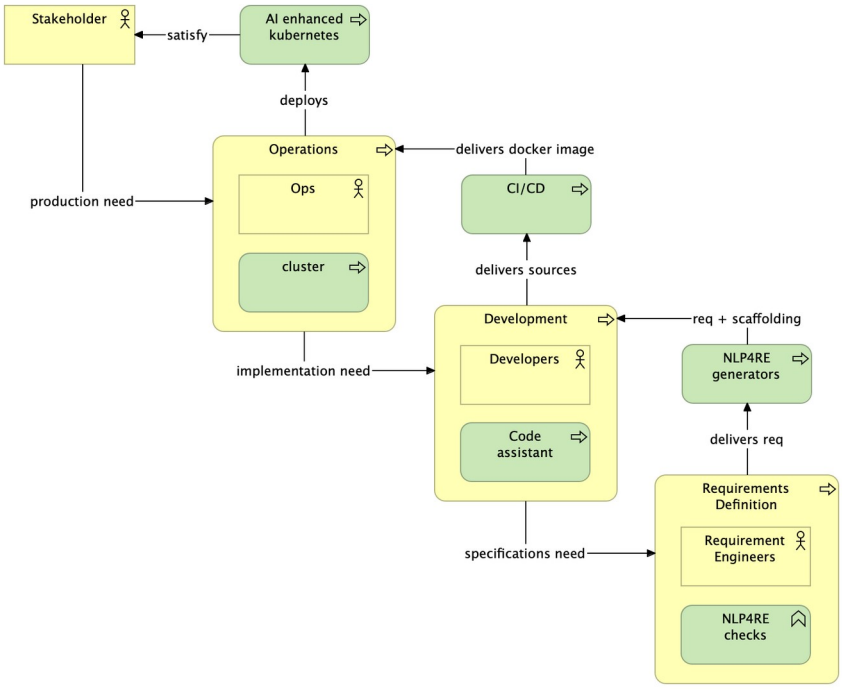
Despite the status of “mission-critical”, many SC’s applications do not run in a Kubernetes cluster and have an availability of 99.5% not in line with modern possibilities. Internal architectural limitations make the progress in this area extremely difficult and slow. Human support is still essential to react as quickly as possible to incidents, but this approach is suboptimal and adds pressure to the teams.

Discussion

In a company like SC, an alternative approach would be a process “in pull” (Spearmen, 1992). Operations would receive the initial signal from stakeholders and they would signal to Developers attaching specifications fitting their infrastructure. Developers would then drive the Analysts in requirement collection. This inversion can help clarify expectations from Operations to Developers and from Developers to Analysts mitigating the need to understand each others’ discipline that is so challenging in DevOps. It would also reduce the chance to bounce artifacts back to a previous step for incompleteness or incompatibilities.

Then it is imaginable the following downstream flow using the mentioned technologies:

- The system validates Analysts' requirements against the template (Arora).
- The system detects RS (Femmer).
- The system analyzes the semantics of requirements (Ferrari).
- The system transforms requirements into UML (Hamza) or directly into code.
- Developers generate code based on templates (Shinde) and schemas.
- Developers work with high-quality requirements, advanced scaffolding, clear specifications from Operations about the target environment (Docker and Kubernetes), and can work with code assistance (Github's Copilot).
- Code is automatically scanned for quality and vulnerabilities (OWASP and SonarQube).
- Ops incorporate the artifact in their AI-enhanced Kubernetes cluster.
- AI can automatically detect anomalies in the log and signal Ops.



Business process and groups (yellow), and technologies (green)

Conclusions

NLP and AI made enormous progress in the latest years and may finally become commercially viable, especially considering that big players can train advanced systems with access to a large amount of data. Practical applications are ready to be integrated into the workflow offering a level of quality unreachable with a traditional approach and can help reduce the pressure on Developers and Operations. Organizations like SC, however, still have to do many steps to reach the necessary maturity, and, for them, the applicability of these solutions is limited by the wide presence of legacy software.

References

Arora, C., Sabetzadeh, M., Briand, L. & Zimmer, F. (2015) 'Automated Checking of Conformance to Requirements Templates Using Natural Language Processing'. *IEEE Transactions On Software Engineering* 41:944–968.

Benton, S., Li, X., Lou, Y., & Zhang, L. (2020) 'On the effectiveness of unified debugging: An extensive study on 16 program repair systems'. *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

Chen, M., et al. (2021) *Evaluating Large Language Models Trained on Code*. Available from: https://arxiv.org/abs/2107.03374 [Accessed on 16 October 2021]

Dawood, O. S. & Sahraoui A. E. K. (2017) 'From requirements engineering to uml using natural language processing–survey study'. *European Journal of Engineering and Technology Research* 2.1: 44-50.

Denger C. & Olsson T. (2005) 'Quality Assurance in Requirements Engineering'. In: Aurum A., Wohlin C. (eds) *Engineering and Managing Software Requirements*. Berlin, Heidelberg: Springer. DOI: https://doi.org/10.1007/3-540-28244-0_8

Elron, R. (2021) 'Automatic Vulnerability Remediation – The Trusted and Secure Road to Developer Happiness'. *OWASP's 20th Anniversary Celebration*. Available from: https://www.youtube.com/watch?v=BHbJN-QKGVE [Accessed on 2021 October 22]

Femmer, H, Fernàndez, D.M, Wagner, S., Eder, S. (2017) 'Rapid quality assurance with requirements smells'. *Journal of Systems and Software* 123: 190-213.

Ferrari, A. (2018) 'Natural language requirements processing: from research to practice'. *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*.

Fowler, M., Beck, K., Roberts, D. & Gamma, E. (1999) Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.

Github (2021) Github Copilot – Your AI pair programmer. Available from: https://copilot.github.com [Accessed on 16 October 2021].

Hemon, A., Lyonnet, B., Rowe, F. & Fitzgerald, B. (2020). From Agile to DevOps: Smart Skills and Collaborations. *Information Systems Frontiers* 22: 927-945. DOI 10.1007/s10796-019-09905-1

Hemon-Hildgen, A., Rowe, F. & Monnier-Senicourt, F. (2020) Orchestrating automation and sharing in DevOps teams: a revelatory case of job satisfaction factors, risk and work conditions. *European Journal of Information Systems* 29(5): 474-499. DOI: 10.1080/0960085X.2020.1782276

Hammond, P., Ahmad, B., Tan, B., Dolan-Gavitt, B. & Karr, R. (2021) *An Empirical Cybersecurity Evaluation of GitHub Copilot's Code Contributions*. Available from: *https://arxiv.org/pdf/2108.09293.pdf* [Accessed on 16 October 2021].

Hamza, Z. A. & Mustafa H. (2019) 'Generating UML use case models from software requirements using natural language processing.' *8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*.

Kassab, M., Neill, C. & Laplante, P. (2014) 'State of practice in requirements engineering: contemporary data'. *Innovations in Systems and Software Engineering* 10: 235-241. DOI: 10.1007/s11334-014-0232-4

Kasser, J.E., Williams, V.R., 'What Do You Mean You Can't Tell Me if My Project is in Trouble?'. *First European Conference on Software Metrics (FESMA 98).* Belgium, Antwerp, 1998.

Kuhn, D. R, Raunak, M. D. & Kacker, R. (2017) ‘An Analysis of Vulnerability Trends, 2008-2016’. *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*.

Kuusinen, K., et al. (2018) ‘A large agile organization on its journey towards DevOps.’ *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*.

Latah, M. & Toker, L. (2019). ‘Artificial intelligence enabled software-defined networking: a comprehensive overview’. *IET networks 8(2):* 79-99.

Laukkanen, E., Itkonen, J. & Lassenius C. (2017) ‘Problems, causes and solutions when adopting continuous delivery—A systematic literature review.’ *Information and Software Technology* 82: 55-79.

Lobo, L. O. & Arthur, J. D., (2005) *Effective Requirements Generation: Synchronizing Early Verification & Validation, Methods and Method Selection Criteria*. USA, Virginia, Virginia Tech Blacksburg, Department of Computer Science.

Merkel, D. (2014) Docker: lightweight linux containers for consistent development and deployment. *Linux journal.* Available from: https://www.seltzer.com/margo/teaching/CS508.19/papers/merkel14.pdf [Accessed on 17 October 2021]

Mich, L. (1996) ‘NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA’. *Natural language engineering.* 2(2): 161-187.

Mich, L., Mariangela, F. & Pierluigi, N. I., (2004) Market research for requirements analysis using linguistic tools. *Requirements Engineering* 9: 40-56. DOI 10.1007/s00766-003-0179-8

Nabi, M., Toeroe M. & Khendek F. (2016) ‘Availability in the cloud: State of the art’. *Journal of Network and Computer Applications.* 60: 54-67.

NIST, National Institute of Standards and Technology (2021) NVD – Data Feeds. Available from: https://nvd.nist.gov/vuln/data-feeds [Accessed on 22 October 2021]

OpenAI (2021) OpenAI Codex Live Demo. Available from: https://www.youtube.com/watch?v=SGUCcjHTmGY [Accessed on 16 October 2021]

OWASP, Open Web Application Security Project (2021) OWASP Dependency Check Project. Available from: https://owasp.org/www-project-dependency-check/ [Accessed on 22 October 2021]

Padman, P., Narlawar, A., Surana, P., Kasliwal, R. & Sonwale, M. (2019) ‘AI Powered System Providing Knowledge Based Solution for Errors in Server Logs’, *5th International Conference on Computing Communication Control and Automation.* Pune, India, 19-21 Sept. 2019, IEEE. DOI: 10.1109/ICCUBEA47591.2019.9129276

Rodriguez, D., Sicilia, M.A., Garcia, E. & Harrison, R. (2012) Empirical findings on team size and productivity in software development. *Journal of Systems and Software* 85(3): 562-570

Schwaber K. & Sutherland J. (2020) The Scrum Guide. Available from: https://www.scrum.org/resources/scrum-guide [Accessed on 17 October 2021]

Shah J. & Dubaria D. (2019) ‘Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform’, *2019 IEEE 9th Annual Computing and Communication*

Workshop and Conference (CCWC) 0184-0189. DOI: 10.1109/CCWC.2019.8666479.

Shinde, K. & Sun, K. (2016) "Template-based code generation framework for data-driven software development." *2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD)*.

Smeds, J., Nybom, K. & Porres I. (2015) 'DevOps: a definition and perceived adoption impediments.' *International conference on agile software development*. Springer, Cham.

SonarSource S.A. (2021) Code Quality and Code Security | SonarQube. Available from: https://www.sonarqube.org/ [Accessed on 22 October 2021]

Spearman, M.L. & Zazanis, M. A. (1992) 'Push and Pull Production Systems: Issues and Comparisons'. *Operations Research* 40(3): 521-532. http://dx.doi.org/10.1287/opre.40.3.521

Wells, D. (2000) Extreme Programming: A gentle introduction. Available from: http://www.extremeprogramming.org/ [Accessed on 17 October 2021]

Splunk Inc. (2021) Splunk | Turn Data Into Doing. Available from https://www.splunk.com/ [Accessed on 22 October 2021]

Zhao L., et al. (2020) 'Natural language processing (NLP) for requirements engineering: A systematic mapping study'. *arXiv preprint arXiv:2004.01099*.