

Investigating the Interplay between Developers and Automation

Omar Elazhary
University of Victoria
omazhary@uvic.ca

Abstract—Continuous practices are a staple of the modern software development workflow. Automation, in particular, is widely adopted due to its benefits related to quality and productivity. However, automation, similarly to all other aspects of the software development workflow, interacts with humans (in this case developers). While some work has investigated the impact of automation on developers, it is not clear to what extent context and process influence that impact. We present our ADEPT theory of developers and automation, in an attempt to bridge this gap and identify the possible ways context, process, and other factors may influence how developers perceive, interpret, and interact with automation.

Index Terms—Software engineering, automation, continuous software development, continuous integration, software engineering theory.

I. INTRODUCTION

Continuous practices are a staple of modern software development [1]. The most commonly adopted continuous practice is automating some tasks of the development workflow. These tasks typically include building an application, testing it, releasing it, and possibly deploying it to a client’s system. Given the benefits to productivity and quality such automation brings [2], [3], it is no wonder such tools have seen widespread adoption.

However, automation in general has had an uneasy relationship with humans [4]–[7]. One phenomenon related to automation is that of misuse, or as it is later called complacency, which denotes the over-reliance on the results produced by automation. The opposite extreme to that phenomenon is disuse which describes a lack of trust in the results produced by automation. Finally, there is automation abuse, which captures automation that is configured in such a way that it hinders the process it is meant to improve.

In the software engineering domain, particularly in the context of continuous practices, such phenomena also occur in addition to new phenomena specific to software engineering. For instance, software developers can be impacted by automated build results, which affects their next commit [8]. Or the negative impact adopting automated build tools has on a project’s developer-base [9].

Unfortunately, the interplay between humans and automated tools has been under-explored in software engineering. We hope to explore this area and map out the various phenomena that relate to the interactions between software developers and the automation they use in their workflows.

II. BACKGROUND AND RELATED WORK

Most of the literature regarding automation in the context of continuous practices is centered around two major areas of research: optimizing the automation workflow, and detecting problems with automation configurations. Optimization efforts typically center around selectively or incrementally [10] executing tests or builds, or exploring how to shorten build durations [11]. Detecting problems with configurations, on the other hand, focuses on documenting different incorrect configuration methods [12], [13], or designing tools that detect such incorrect configurations [14].

More recently, however, there has been some work on the human aspects of continuous practices. As mentioned in section I, some studies investigated the relationship between failing builds and developer sentiment over time [8]. Souza and Silva observe that failing builds relate to negative commit messages when trying to fix the error. And negative commit messages are more likely to produce failing builds. Another example of automation having an impact on human behavior is the work done by Gupta et al. [9], where they observe that a project’s ability to retain and attract developers decreases upon adopting Travis CI, a popular continuous integration tool. However, it is not clear *why* these phenomena occur.

III. PREVIOUS STUDIES

To map out the different relationships between developers and automation, we started by examining open source projects and how automation was used within that context. We examined the contribution guidelines for several open source projects to examine how newcomers were being instructed to use and interpret the automation that was attached to the project repositories [15]. Surprisingly, we found that none of the projects discussed their attached automation tools beyond them being used for testing purposes.

We then decided to investigate continuous practices in an industrial context, and how they might impact development workflow. To that end, we conducted two studies; the first explored the impact of continuous practices on non-functional requirements [16], and the second focused on the perceived benefits and costs of continuous practices [17]. In the former study, we found that the automation brought about by continuous practices had facilitated dealing with particular non-functional requirements (such as configurability) with developers reprioritizing how they deal with these requirements. In the second study, we observed that the benefits

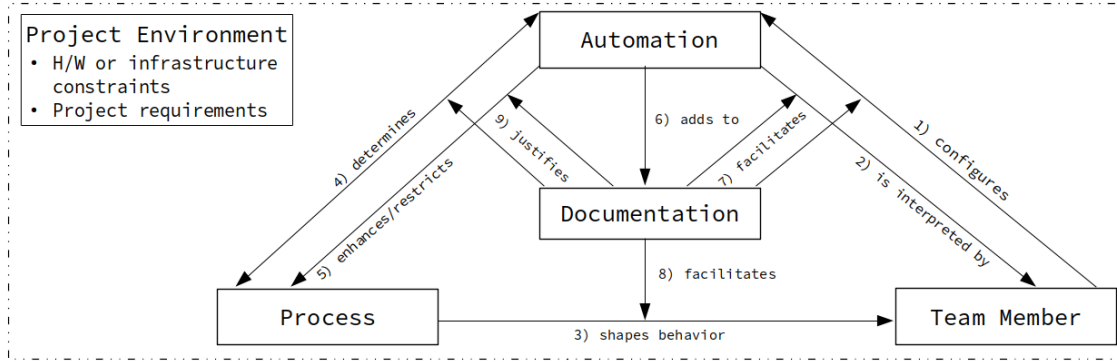


Fig. 1. Visual Representation of the ADEPT Theory (Boxes are constructs, Edges are relationships)

and costs brought about by continuous practices (including automation) are a result of the practices, as opposed to the overarching practice set. For instance, automation was perceived to increase development cycle velocity, but—without tests—it had no impact on quality. The association of costs and benefits with their finer-grained practices (as opposed to the overarching methodology of continuous integration) helped us better map the relationship between developers and automation. The mapping, in turn, led to our most recent study.

IV. PROPOSED SOLUTION AND RESULTS

To gain a better understanding of how developers interact with automation in their specific contexts, we decided to develop a theory. We considered this a step towards a more theory-oriented software engineering research field as argued by Stol and Fitzgerald [18]. To the best of our knowledge at the time, there was no theory explaining the different ways developers interacted with automation.

Based on our previous studies [15], [17], we attempted to create an explanatory theory to better represent the phenomena we were observing and to provide a starting point to generate hypotheses about human-centric phenomena related to automation [19]. The ADEPT theory (**A**utomation, **D**ocumentation, **P**roject **E**nvironment, **P**rocess, **T**eam **M**ember) maps the relationships we observed and sets the scene for further investigation. A representation of ADEPT can be seen in Figure 1.

Team member refers to people who interact with automation, be they developers, program managers, testers, or others. **Automation** is abstracted to include automated tools that team members interact with throughout their development process. **Process** encapsulates the various practices and activities that constitute their development workflow. **Documentation** represents the artifacts with which developers communicate knowledge about their process, tools, and code.

The edges between the constructs represent relationships we observed during our previous studies [17]. Two relationships stand out in particular; automation adds to documentation, and the bidirectional relationship between automation and process. We found that automation serves to document tacit build knowledge in form of configuration and build scripts. We also found that the desire to implement a particular process impacts

the choice of automation used. We also hypothesize that the reverse is also true in that the desire to use a particular tool can force developers to implement a different process.

The theory captures context as available through both the project environment and the process within which developers operate. In fact, it illustrates that these relationships may only produce the results discussed by literature under a constant context, which is not the case. We also found that the relationship between developers and automation is impacted by documentation. In many cases, developers would resort to a tool’s documentation to either run an automated build manually, or interpret a tool’s results.

Furthermore, our theory allows for the formation of compound relationships. For instance, while a developer can use documentation to interpret the results of automation, that behavior is shaped by the process within which they operate. Without the process priming that particular behavior, developers could interpret automation results differently.

V. FUTURE WORK

While our previous studies provided us with an understanding of possible relationships between developers and automation, we need to operationalize the concepts our theory proposes. And we still do not know if these relationships apply in different contexts. We make the argument that context is important, and the results should always be interpreted within it, however, operationalizing the context is not as straightforward as it seems.

Currently, we plan to investigate how best to operationalize these concepts, and test the relationships we posit in our theory. We plan to investigate if the relationships can be used to interpret previous results in both industry and open source contexts to contrast these differences. The main objective is to determine to what extent our theory describes the relationship between automation and developers, and whether it can be improved.

Similarly to our previous studies, we plan to use a mixed-methods approach to data collection. We plan to conduct online surveys that help us capture context and human-centric aspects of the relationships we observed. Additionally, we plan to mine repository development logs to both triangulate the survey results, and objectively observe development metrics and relate them to automation properties.

REFERENCES

- [1] B. Fitzgerald and K.-J. Stol, “Continuous software engineering: A roadmap and agenda,” *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017.
- [2] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, “Quality and productivity outcomes relating to continuous integration in github,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 805–816.
- [3] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, “Usage, costs, and benefits of continuous integration in open-source projects,” in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 426–437.
- [4] R. Parasuraman, R. Molloy, and I. L. Singh, “Performance consequences of automation-induced complacency,” *The International Journal of Aviation Psychology*, vol. 3, no. 1, pp. 1–23, 1993.
- [5] R. Parasuraman and D. H. Manzey, “Complacency and bias in human use of automation: An attentional integration,” *Human factors*, vol. 52, no. 3, pp. 381–410, 2010.
- [6] I. L. Singh, R. Molloy, and R. Parasuraman, “Automation-induced complacency”: Development of the complacency-potential rating scale,” *The International Journal of Aviation Psychology*, vol. 3, no. 2, pp. 111–122, 1993.
- [7] R. Parasuraman and V. Riley, “Humans and automation: Use, misuse, disuse, abuse,” *Human factors*, vol. 39, no. 2, pp. 230–253, 1997.
- [8] R. Souza and B. Silva, “Sentiment analysis of travis ci builds,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 459–462.
- [9] Y. Gupta, Y. Khan, K. Gallaba, and S. McIntosh, “The impact of the adoption of continuous integration on developer attraction and retention,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 491–494.
- [10] G. Maudoux and K. Mens, “Bringing incremental builds to continuous integration,” in *Proc. 10th Seminar Series Advanced Techniques & Tools for Software Evolution*, 2017.
- [11] T. A. Ghaleb, D. A. Da Costa, and Y. Zou, “An empirical study of the long duration of continuous integration builds,” *Empirical Software Engineering*, vol. 24, no. 4, pp. 2102–2139, 2019.
- [12] K. Gallaba and S. McIntosh, “Use and misuse of continuous integration features: An empirical study of projects that (mis) use travis ci,” *IEEE Transactions on Software Engineering*, 2018.
- [13] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall, and M. Di Penta, “An empirical characterization of bad practices in continuous integration,” *Empirical Software Engineering*, vol. 25, no. 2, pp. 1095–1135, 2020.
- [14] C. Vassallo, S. Proksch, H. C. Gall, and M. Di Penta, “Automated reporting of anti-patterns and decay in continuous integration,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 105–115.
- [15] O. Elazhary, M.-A. Storey, N. Ernst, and A. Zaidman, “Do as i do, not as i say: Do contribution guidelines match the github contribution process?” in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 286–290.
- [16] C. Werner, Z. S. Li, D. Lowlind, O. Elazhary, N. A. Ernst, and D. Damian, “Continuously managing nfes: Opportunities and challenges in practice,” *IEEE Transactions on Software Engineering*, 2021.
- [17] O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. A. Ernst, and M.-A. Storey, “Uncovering the benefits and challenges of continuous integration practices,” *IEEE Transactions on Software Engineering*, 2021.
- [18] K.-J. Stol and B. Fitzgerald, “Theory-oriented software engineering,” *Science of Computer Programming*, vol. 101, pp. 79–98, 2015.
- [19] O. Elazhary, M.-A. Storey, N. A. Ernst, and E. Paradis, “Adept: A socio-technical theory of continuous integration,” in *43rd International Conference on Software Engineering*. IEEE, 2021.