UNIVERSIDADE FEDERAL DO ABC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Daniel D'Angelo Resende Barros

ANALISANDO O COMPORTAMENTO DA PERFORMANCE NA ENTREGA DE SOFTWARE EM PROJETOS OPEN SOURCE EM TIMELINE DE RELEASE POR MEIO DE MÉTRICAS DE ENTREGA

Santo André, SP

2023

DANIEL D'ANGELO RESENDE BARROS

ANALISANDO O COMPORTAMENTO DA PERFORMANCE NA ENTREGA DE SOFTWARE EM PROJETOS OPEN SOURCE EM TIMELINE DE RELEASE POR MEIO DE MÉTRICAS DE ENTREGA

Dissertação de Mestrado apresentada ao Curso de Pós-Graduação em Ciência da Computação (Área de concentração: Sistemas de Computação) da Universidade Federal do ABC como requisito parcial para obtenção do Título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Flávio Eduardo Aoki Horita
Coorientador: Prof. Dr. Igor Scaliante Wiese

Santo André

2023

Sistema de Bibliotecas da Universidade Federal do ABC
Elaborada pelo Sistema de Geração de Ficha Catalográfica da UFABC
com os dados fornecidos pelo(a) autor(a).

D'Angelo Resende Barros, Daniel
Analisando o comportamento da performance na entrega de software em projetos open source em timeline de release por meio de métricas de entrega / Daniel D'Angelo Resende Barros. — 2023.

111 fls.

Orientador: Flávio Eduardo Aoki Horita
Coorientador: Igor Scaliante Wiese

Dissertação (Mestrado) — Universidade Federal do ABC, Programa de Pós-Graduação em Ciência da Computação, Santo André, 2023.

1. Desenvolvimento de software. 2. Performance na Entrega de Software. 3. Projetos de Software de Código Aberto. 4. Timeline de Release. 5. Métricas de Entrega. I. Aoki Horita, Flávio Eduardo. II. Scaliante Wiese, Igor. III. Programa de Pós-Graduação em Ciência da Computação, 2023. IV. Título.

Este exemplar foi revisado e alterado à versão original, de acordo com as observações levantadas pela banca no dia da defesa, sob responsabilidade única do (a) autor(a) e com a anuência do(a) orientador(a).

Santo André/SP, 28 **de** Junho **de** 2023.

Assinatura do(a) autor(a): ___

Documento assinado digitalmente
gov.br
DANIEL D ANGELO RESENDE BARROS
Data: 28/06/2023 19:12:14-0300
Verifique em https://validar.iti.gov.br

Assinatura do(a) orientador(a): __

Documento assinado digitalmente
gov.br
FLAVIO EDUARDO AOKI HORITA
Data: 28/06/2023 19:05:05-0300
Verifique em https://validar.iti.gov.br

MINISTÉRIO DA EDUCAÇÃO
Fundação Universidade Federal do ABC
Avenida dos Estados, 5001 – Bairro Santa Terezinha – Santo André – SP
CEP 09210-580 · Fone: (11) 4996-0017

Ata de Defesa de Dissertação de Mestrado

No dia 14 de Abril de 2023 às 09h45, https://conferenciaweb.rnp.br/events/daniel-barros, realizou-se a Defesa de Dissertação de Mestrado, que constou da apresentação do trabalho intitulado **"Analisando o comportamento da performance na entrega de software em projetos open source em timeline de release por meio de métricas de entrega "** de autoria do candidato, **DANIEL D ANGELO RESENDE BARROS**, RA nº 21201931116, discente do Programa de Pós-Graduação em CIÊNCIA DA COMPUTAÇÃO da UFABC. Concluídos os trabalhos de apresentação e arguição, o candidato foi considerado Aprovado pela Banca Examinadora.
E, para constar, foi lavrada a presente ata, que vai assinada pelos membros da Banca.

Documento assinado digitalmente
gov.br PAULO ROBERTO MIRANDA MEIRELLES
Data: 17/04/2023 10:57:22-0300
Verifique em https://validar.iti.gov.br

Prof.(a) PAULO ROBERTO MIRANDA MEIRELLES
UNIVERSIDADE FEDERAL DO ABC - Membro Titular

Documento assinado digitalmente
gov.br UIRA KULESZA
Data: 12/06/2023 10:10:01-0300
Verifique em https://validar.iti.gov.br

Prof.(a) UIRA KULESZA
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE - Membro Titular

Prof.(a) CARLOS ALBERTO KAMIENSKI
UNIVERSIDADE FEDERAL DO ABC - Membro Suplente

Prof.(a) DAVI VIANA DOS SANTOS
UNIVERSIDADE FEDERAL DO MARANHÃO - Membro Suplente

Documento assinado digitalmente
gov.br FLAVIO EDUARDO AOKI HORITA
Data: 17/04/2023 08:13:11-0300
Verifique em https://validar.iti.gov.br

Prof.(a) FLAVIO EDUARDO AOKI HORITA
UNIVERSIDADE FEDERAL DO ABC - Presidente

* Por ausência do membro titular, foi substituído pelo membro suplente descrito acima: nome completo, instituição e assinatura

MINISTÉRIO DA EDUCAÇÃO
Fundação Universidade Federal do ABC
Avenida dos Estados, 5001 – Bairro Santa Terezinha – Santo André – SP
CEP 09210-580 · Fone: (11) 4996-0017

Ressalvas e sugestões da Banca examinadora:

Conforme Resolução da CPG nº 05, de 26 de abril de 2016, os membros que participaram de modo remoto, pelos quais o Presidente da Banca assinou no anverso desta Ata, foram:

Flavio Horita, Paulo Meirelles, Uirá Kulesza

Por sugestão da Banca Examinadora, o novo título passa a ser (em letra de forma e legível):

Assinatura do Presidente da Banca (apenas em caso de alteração)

* Por ausência do membro titular, foi substituído pelo membro suplente descrito acima: nome completo, instituição e assinatura

MINISTÉRIO DA EDUCAÇÃO
Fundação Universidade Federal do ABC
Avenida dos Estados, 5001 – Bairro Santa Terezinha – Santo André – SP
CEP 09210-580 · Fone: (11) 4996-0017

FOLHA DE ASSINATURAS

Assinaturas dos membros da Banca Examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato, DANIEL D ANGELO RESENDE BARROS realizada em 14 de Abril de 2023:

Documento assinado digitalmente
gov.br PAULO ROBERTO MIRANDA MEIRELLES
Data: 17/04/2023 11:01:34-0300
Verifique em https://validar.iti.gov.br

Prof.(a) PAULO ROBERTO MIRANDA MEIRELLES
UNIVERSIDADE FEDERAL DO ABC

Documento assinado digitalmente
gov.br UIRA KULESZA
Data: 12/06/2023 10:12:10-0300
Verifique em https://validar.iti.gov.br

Prof.(a) UIRA KULESZA
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Prof.(a) CARLOS ALBERTO KAMIENSKI
UNIVERSIDADE FEDERAL DO ABC

Prof.(a) DAVI VIANA DOS SANTOS
UNIVERSIDADE FEDERAL DO MARANHÃO

Documento assinado digitalmente
gov.br FLAVIO EDUARDO AOKI HORITA
Data: 17/04/2023 08:11:01-0300
Verifique em https://validar.iti.gov.br

Prof.(a) FLAVIO EDUARDO AOKI HORITA
UNIVERSIDADE FEDERAL DO ABC - Presidente

* Por ausência do membro titular, foi substituído pelo membro suplente descrito acima: nome completo, instituição e assinatura

To Fabiana, the love of my life
To Rafael and Pietro, my two reasons to live
To Márcia, mother and intellectual mentor
To José Eduardo, father and supporter of my education (in memoriam)

Para Fabiana, o amor da minha vida
Para Rafael e Pietro, minhas duas razões de viver
Para Márcia, mãe e mentora intelectual
Para José Eduardo, pai e incentivador aos meus estudos (in memoriam)

Citation for Financial Support

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

Acknowledgments

First of all, I would like to say that I feel very lucky to have completed this valuable step in my life. In a country where less than 1% of people complete a master's degree (the average in countries where science is valued exceeds 10%) I must feel privileged. This journey provided me with the most valuable legacy I could acquire: the intellectual legacy. Being able to publish academic works, and developing scientific thinking were experiences that I will never forget in my life. I am very proud to say that today I can be called a computer scientist.

All of this would not have been possible if it weren't for the teachers who have passed through my life. Firstly, my father and mother, teachers of life who always taught me that study was the path to the main achievements in life. My father was a great encourager and motivator of my studies, and my mother was an intellectual mentor and example of dedication to academic life. My sisters Flávia and Fernanda, companionship teachers shared with me the first lessons in this life. Professor Silvia Regina, my Elementary School teacher at Saint Exupéry School, tenderly and firmly managed to show me the great beauty and power of mathematics. Professors at ETFSP (today IFSP) and PUC-SP contributed to the beginning of my passion for Computing. My dear wife Fabiana, a teacher of love and compassion who always believed in me, even when I lost confidence in myself. My children Rafael and Pietro, special kids that teach joy, love, and new learnings every day that only the purity of children can pass on. Professors Flávio Horita and Igor Wiese, Master's advisors, and great friends. And finally, all the friends, family, and teachers that have passed through my life and that I will not be able to mention here. I was taught that life is a great learning experience. Thank you very much to all of you!

São Paulo, February 2023
A rainy Carnival Sunday

Agradecimentos

Primeiramente gostaria de dizer que me sinto uma pessoa de muita sorte por concluir essa valiosa etapa na minha vida. Em um país onde menos de 1% das pessoas concluem o mestrado (a média em países que a ciência é valorizada ultrapassa 10%) não tenho como não agradecer e me sentir privilegiado. Essa jornada proporcionou-me o legado mais valioso que eu poderia adquirir: o legado intelectual. Poder publicar trabalhos acadêmicos, desenvolver o pensamento científico foram experiências que jamais esquecerei na minha vida. Tenho muito orgulho de dizer que hoje posso ser chamado de cientista da computação.

Tudo isso não seria possível se não fossem os professores que passaram pela minha vida. Primeiramente meu pai e minha mãe, professores da vida e que sempre me ensinaram que o estudo era o caminho para as principais conquistas da vida. Meu pai como um grande incentivador e motivador dos meus estudos, minha mãe como mentora intelectual e exemplo de dedicação à vida acadêmica. Minhas irmãs Flávia e Fernanda, professoras de companherismo que compartilharam comigo as primeiras lições nessa vida. Professora Silvia Regina, minha professora no Ensino Fundamental na Escola Saint Exupéry que conseguiu com ternura e firmeza me mostrar a grande beleza e poder da matemática. Professores da ETFSP (hoje IFSP) e PUC-SP que contribuíram para o início dessa minha paixão com a Computação. Minha querida esposa Fabiana, professora de amor e compaixão que sempre acreditou em mim, mesmo nos períodos que eu perdi a confiança em mim mesmo. Meus filhos Rafael e Pietro, professores que todo dia ensinam alegria, amor e novos aprendizados que somente a pureza das crianças consegue passar. Professores Flávio Horita e Igor Wiese, orientadores de mestrado e grandes amigos. E por fim, todos os amigos, familiares e professores que passaram pela minha vida e que não conseguirei citar por aqui. Me ensinaram que a vida é um grande aprendizado. Muito obrigado a todos vocês!

São Paulo, Fevereiro de 2023
Um domingo chuvoso de Carnaval

"What is essential is invisible to the eyes."

—Antoine de Saint-Exupéry

Resumo

O desenvolvimento de software tornou-se um dos principais fatores para a entrega de serviços hoje. Há uma demanda crescente por entrega de software mais rápida, chamada aqui de Release. Surgiram métodos de desenvolvimento ágil que estão ajudando a acelerar a entrega de software. Assim, o desempenho de entrega de software melhorou em termos de frequência convergindo para ciclos de lançamento rápidos que podem reduzir o tempo de lançamento. No entanto, apenas usar ciclos rápidos pode não ser suficiente, pois medir a entrega de software levanta algumas questões importantes, como está ocorrendo o processo de entrega de software e como deveria ser. Surgiram várias estratégias para medir a entrega de software, incluindo Software Delivery Performance (SDP), em que a entrega de software é estimada em termos de capacidades. Popularidade em Projetos de Software de Código Aberto (OSSP) significa que um projeto é suficientemente reconhecido na comunidade para atender à demanda e, portanto, provavelmente estará pronto para ser medido por meio de uma abordagem de entrega de software como o SDP. Diante disso, este trabalho oferece maneiras de analisar o comportamento do SDP em OSSP populares em uma linha do tempo de Release por meio de métricas na logística de entrega. Os resultados demonstraram que a popularidade é uma abordagem eficiente, aumentando a confiabilidade e a precisão do trabalho. Conclui-se que a produtividade do desenvolvedor melhora o SDP, e o valor deste trabalho é que ele é capaz de analisar o comportamento do SDP no OSSP em uma linha do tempo de lançamento por meio de métricas de entrega de maneira automatizada. O código-fonte que implementa a metodologia é publicado como um pacote de replicação, e o conjunto de dados final com entradas e saídas está disponível para incentivar a reprodutibilidade e pesquisas futuras.

Palavras-chave: Desenvolvimento de software, Performance na Entrega de Software, Projetos de Software de Código Aberto, Timeline de Release, Métricas de Entrega.

Abstract

Software development has become one of the main factors for service delivery today. There is an increasing demand for faster software delivery and more assertive communication called here Release. Agile development methods have emerged which are helping to accelerate software delivery. As a result, software delivery performance has improved in terms of frequency and led to more adopters of rapid release cycles which can reduce time-to-market. However, just using rapid releases may not be enough as measuring software delivery raises some key questions, like how is the software delivery process taking place and what it should be like. A number of strategies for measuring software delivery have appeared such as Software Delivery Performance (SDP) where software delivery is estimated in terms of evolving capabilities. Popularity in Open Source Software Projects (OSSP) means that a project is sufficiently recognized in the community to meet the demand for software and, is thus likely to be ready to be measured through a software delivery approach like SDP. In light of this, this work offers a means of analyzing SDP behavior in popular OSSP on a Release timeline basis through metrics in delivery logistics. The results demonstrated that popularity is an efficient filtering approach, as it improves the OSSP delivery, by enhancing the work's reliability and accuracy. It can be concluded that developer productivity improves SDP, and the value of this work is that it is able to analyze SDP behavior in OSSP on a Release timeline basis through delivery metrics in an automated manner. The source code that implements the methodology is published as a replication package, and the final dataset with inputs and outputs is available to encourage reproducibility and future research.

Keywords: Software development, Software Delivery Performance, Open Source Software Projects, Release Timeline, Delivery Metrics.

List of Figures

Figure 1 – Research Design: Connection between the RQ, Research Hypothesis, the overall objective of this work, the research SO, methods, and expected outcomes . . . 9
Figure 2 – The Nexus Research Initiative foundations topics . . . 11

Figure 3 – Framework for CSE proposed by Barcellos (M.Barcellos, 2020) . . . 17
Figure 4 – DORA platform proposed by Forsgren et al.(N.Forsgren et al., 2017) . . . 19
Figure 5 – Framework for SDP proposed by Forsgren et al.(N.Forsgren; J.Humble; G.Kim, 2018) in the last year of the research . . . 20

Figure 6 – The SPACE framework proposed by Forsgren et al.(N.Forsgren et al., 2021) 26

Figure 7 – gthbmining.rc Architecture (D.Barros; F.Horita; D.Fantinato, 2020) . . . 31
Figure 8 – DevOps framework in a cyber-physical system case (D.Barros; F.Horita, 2020b) . . . 32
Figure 9 – REP supporting CDE, and how it is improved through RC (D.Barros; F.Horita, 2020a) . . . 33
Figure 10 – MSR Extended CookBook High-level Themes (D.Barros et al., 2021) . . . 34
Figure 11 – POC used Languages . . . 35
Figure 12 – POC SDP Metrics Box Plot . . . 36
Figure 13 – POC OSSP SDP Taxonomy trend . . . 37
Figure 14 – POC OSSP CSE Framework trend . . . 39

Figure 15 – Methodology proposed . . . 44
Figure 16 – OSSP GitHub data structure . . . 45
Figure 17 – Methodology Entity Relationship Diagram . . . 47

Figure 18 – Software Development Languages used . . . 55
Figure 19 – SDP Metrics Box Plot . . . 57
Figure 20 – SDP Metrics Box Plot Top 5 Languages . . . 58
Figure 21 – SPACE Framework mapped to this work metrics . . . 65

List of Tables

Table 1 – Data Extraction classifying data in the POC 36
Table 2 – Data Extraction popularity data in the POC 37
Table 3 – Data Extraction classifying data in the POC 38
Table 4 – Data Extraction popularity data in the POC 38

Table 5 – Software Development Languages used 54
Table 6 – Percentage Count of Releases considering CI/CD 56
Table 7 – Shapiro-Wilk normality test considering CI/CD 59
Table 8 – Shapiro-Wilk normality test considering CI/CD Top 5 Languages 59
Table 9 – Wilcoxon signed rank test with continuity correction 60
Table 10 – Wilcoxon signed rank test with continuity correction Top 5 Languages . . 60
Table 11 – Cliff's Delta 61
Table 12 – Cliff's Delta Top 5 Languages 61

List of abbreviations and acronyms

AI Artificial Intelligence.

ASS Applied Software System.

BAU Business as Usual.

CD Continuous Deployment.

CDE Continuous Delivery.

CE Continuous Experimentation.

CI Continuous Integration.

CMCC Center for Mathematics, Computing and Cognition.

CSE Continuous Software Engineering.

CSyE Complex System Engineering.

DORA DevOps Research and Assessment.

DT Digital Transformation.

MRS Mining Software Repositories.

MTTR Mean Time to Restore.

OSSP Open Source Software Projects.

POC Proof of Concept.

RC Release Candidates.

REP Release Engineering Pipeline.

RQ Research question.

SBT Society Business Transformation.

SDP Software Delivery Performance.

SO Specific objectives.

SoS Systems of Systems.

UFABC Universidade Federal do ABC.

XP Extreme Programming.

Contents

1 Introduction **1**

1.1 Motivation and Research Problem . . . 5

1.2 Research Question . . . 7

1.3 Research Hypothesis . . . 7

1.4 Research Objectives . . . 8

1.5 Research Design . . . 9

1.6 Research Context . . . 10

1.7 Organization . . . 12

2 Background **15**

2.1 Initial Considerations . . . 15

2.2 CSE . . . 15

2.3 SDP . . . 17

2.4 Final Considerations . . . 21

3 Related Works **23**

3.1 Initial Considerations . . . 23

3.2 Related Works . . . 23

3.3 Final Considerations . . . 27

4 Literature review and Preliminary results **29**

4.1 Initial Considerations . . . 29

4.2 Scope and Goal . . . 29

4.2.1 Literature Review . . . 30

4.2.2 POC . . . 34

4.3 Final Considerations . . . 39

5 Methodology **41**

5.1 Initial Considerations . . . 41

5.2 Metrics Chosen . . . 41

5.3 Definition . . . 43

5.3.1 Data Extraction . . . 44

5.3.2 Data Analysis . . . 47
5.4 Final Considerations . . . 52

6 Results and discussions **53**
6.1 Initial Considerations . . . 53
6.2 Results . . . 53
6.3 Discussions . . . 61
6.3.1 Implications for literature and practice . . . 62
6.3.2 RQ analysis: Considering Software Delivery Performance as a software delivery measurement model, how do popular Open Source Software Projects behave? . . . 63
6.3.3 Developer Productivity improves SDP . . . 64
6.4 Final Considerations . . . 65

7 Conclusions and future works **67**
7.1 Initial Considerations . . . 67
7.2 Research contributions . . . 67
7.3 Threats to validity . . . 68
7.3.1 Internal Validity . . . 68
7.3.2 External Validity . . . 69
7.4 Future works . . . 69

A Publishing, Participation and Academic Services **81**

B Extra tooling mapped **85**

Appendices **81**

CHAPTER 1

Introduction

Software delivery has become a growing trend in the last few years (S.Stavru, 2014), where everybody is trying to deliver quick and assertive products such as software to reach as many consumers as possible – by means of a website, mobile app, or even a BackOffice product. Software development has thus been facing a challenging environment with uncertain requirements, requests for short releases, and wild competition. Some of these difficulties are caused by the lack of software tasks such as planning, building, and releasing (B.Fitzgerald; K.Stol, 2017).

In an effort to address these challenges, agile development methods (like Extreme Programming (XP) and BizDev) have been used to provide users with flexibility, effectiveness, and agile deliverables (N.Fogelström et al., 2010). XP recommended improvements like Continuous Integration (CI) where software development and delivery would be inter-dependent and synchronous tasks (Beck, 2000). BizDev advised that continuous tasks should be carried out not only with software processes but also with connecting software to achieve the strategic goals of the companies (B.Fitzgerald; K.Stol, 2017).

In parallel, in February 2001 the Manifesto for Agile Software Development, called the "Agile Manifesto" (M.Beedle et al., 2001), was published to describe the ideas and similarities between the agile development methods (P.Abrahamsson et al., 2003; D.Cohen; M.Lindval; P.Costa, 2004). Since then, the Agile Manifesto has embraced everyone who wants to deliver software and stated that software delivery (through its first principle of Continuous Delivery

(CDE)) can be achieved in a disciplined manner by building automated software systems, testing, and deployment strategies to satisfy customers (J.Humble; D.Farley, 2010).

Although the steps required to deliver software more quickly can be challenging and complex (M.Shahin et al., 2017), they are likely to be very rewarding; for example, Neely and Stolt (S.Neely; S.Stolt, 2013) reported the experience of a software company that progressed from a painful eight-week release journey to achieve a model of frequent releases using Kanban. The final result showed power and flexibility with regard to releases, which involved tips for reducing bugs, fewer business-as-usual working hours, and greater team confidence.

As a result, this ability to be agile has demonstrated quality and consistency (M.Michlmayr; B.Fitzgerald; K.Stol, 2015), and hence delivering software as a service has become one of the main strategies adopted in software projects (P.Rodríguez et al., 2017). There has been an increase in the frequency of software delivery which has improved in the last few years and led to more and more adopters of rapid release cycles, including Open Source Software Projects (OSSP) (M.Mäntylä et al., 2015). In other words, the goal has been to reduce time-to-market (D.Costa et al., 2016) by releasing software all the time (E.Laukkanen et al., 2016).

However, speeding up software delivery through rapid releases may not be the only means of addressing all the delivery challenges (D.Costa et al., 2016), as rapid releases might have a negative effect on some aspects of software quality (e.g. a rapid release can delay bug handling (Z.Khalil et al., 2021)). Still, understanding the challenges faced by software projects, whether they involve social or technical issues, is also essential for selecting the right pathway for the evolution of software delivery (Claps; Berntsson Svensson; Aurum, 2015). Moreover, continuous planning, development, delivery, and evaluation are needed to improve software delivery, as well as make effective decisions (M.Barcellos, 2020). Measuring is needed to understand how software delivery takes place, and what steps are required to achieve maturity or enhance capabilities (Humble; Molesky, 2011; N.Forsgren; J.Humble; G.Kim, 2018; M.Barcellos, 2020).

When seeking to understand software delivery and the steps needed to ensure progress, it is worth examining some of the emerging models based on maturity. Continuous Software Engineering (CSE) regards software development as being more than an isolated activity

carried out by different teams. Instead, CSE suggests that software development should be performed by connected teams in a continuous movement as an evolving pathway to maturity through related activities like software planning, building, operational measures, and evaluation (M.Barcellos, 2020).

Maturity models alone might not be sufficient to measure software delivery, as they can experience failures with regard to completeness, correctness, or consistency (O.Ozcan-Top; O.Demirörs, 2013). Rather than only being designed to arrive at a mature state, or apply a linear formula, agile maturity should be prepared to foster subjective capabilities like collaboration, communication, and commitment (R.Fontana et al., 2014). In view of this, some studies have argued that software delivery should include capabilities such as technical practices, uncoupled architectures, lean management practices, and reliable organizational culture. However, formal assessments like CMMI (M.Chrissis; M.Konrad; S.Shrum, 2011) which are designed to detect these capabilities, are not straightforward, as well as being hard to implement, not expandable, subject to bias from the interviewees, and generally restricted to local data (N.Forsgren et al., 2017).

The DORA (DevOps Research and Assessment) platform was created by the application of DevOps,(a combination of the words “development” and “operations”) and recommended practices that could help companies to provide services in response to the needs of customers, through frequent and automated software releases (L.Lwakatare et al., 2016). It was one of the first proposals to assess and measure software delivery through capabilities. With the aid of the client DevOps Team, DORA conducted surveys to invite and collect responses, by adding analytical results to the data received and then creating management reports available on a website (N.Forsgren et al., 2017). Rather than being simply technical or tooling devices, capabilities can be flexible enough to adapt to the transient moment to understand whether a company is following the right path (e.g. Is it worth adopting DevOps?). Going beyond the DORA Platform, Forsgren et al. (N.Forsgren; J.Humble; G.Kim, 2018) compiled several surveys for software companies through a concept called Software Delivery Performance (SDP), which described journeys from the source code that were committed to production. SDP creates a taxonomy to classify software companies in a chart ranging from High Performers

to Low Performers, by means of metrics that rely on capabilities - not on maturity.

Several studies have made use of software delivery measurements (C.Vassallo et al., 2016; C.Vassallo et al., 2017; C.Vassallo et al., 2018; E.Kula et al., 2019; L.Lwakatare et al., 2019), including OSSP (Y.Yu et al., 2016; M.Hilton et al., 2016; Y.Jiang, 2016; F.Zampetti et al., 2017; J.Bernardo; D.Costa; U.Kulesza, 2018; Z.Khalil et al., 2021). More specifically, in OSSP it is known that this has led to widespread benefits like the encouragement of research through transparency, economic and social progress, engagement, and innovation (G.von Krogh; S.Spaeth, 2007). By means of Mining Software Repositories (MSR - techniques that are used to make improvements and innovations in a software project (D.Barros et al., 2021)), OSSP have been also playing a key role in improving software delivery. For example, by using MSR in OSSP it is possible to study the effects of adopting CI in the delivery time of merged Pull Requests (J.Bernardo; D.Costa; U.Kulesza, 2018). Moreover, it has been proved that OSSP, (particularly when hosted in GitHub), makes a maximum effort to collect data through MSR and provide information about software delivery (e.g. Barros et al. (D.Barros; F.Horita; D.Fantinato, 2020); it has also published a tool to discover the DevOps trends in GitHub OSSP).

Popularity in OSSP provides key information to show the open-source community whether the project is experiencing growing acceptance, satisfying users and contributors, and meeting the users' expectations (H.Borges; A.Hora; M.Valente, 2016). In view of this, OSSP popularity does not just mean that the project is known or admired; it means that the project is definitely used by several users (e.g. the most popular OSSP found here *freeCodeCamp/freeCodeCamp*, which is an open source educational software) and claims to have millions of people in many places learning a code together [1]). This is done by complying with the software requirements displayed here, like competitiveness, dynamic environments, and a constant demand for more releases. This means that popular OSSP should be able to be measured through a software delivery approach.

[1] <https://www.freecodecamp.org/news/about>

1.1 Motivation and Research Problem

Current global competitiveness has been leading many software projects to accelerate their delivery process, instead of working several days on a major new release (M.Mäntylä et al., 2015). The concept of rapid releases (F.Khomh et al., 2012) has changed the way modern applications, like Google Chrome (B.Adams; S.McIntosh, 2016) and Spotify (H.Kniberg, 2014), develop and deliver software. Although the benefits of rapid releases have been generally validated (e.g. by reduced time-to-market (F.Khomh et al., 2012), easier release planning with smaller scope (Beck, 2000), faster functionalities, and security updates for users (M.Mäntylä et al., 2015)), rapid releases can have a harmful effect on software quality by delaying teamwork. This is owing to infrastructural problems or testing dependencies, a lower average complexity for coding, and a feeling by developers that their decisions on implementations decisions and design are harmed (E.Kula et al., 2019).

The possible negative aspects of rapid releases can also affect OSSP by making the software less reliable (F.Khomh et al., 2012; D.Costa et al., 2016), and causing a large number of technical failures (M.Tufano et al., 2015), as well as adding more time pressure on teams (J.Rubin; M.Rinard, 2016). Thus, rather than only adopting rapid releases, it is important to collect and evaluate the areas that matter to software delivery (E.Kula et al., 2019). In other words, it is essential to measure software delivery, by recognizing the current situation, as well as having a defined evolving pathway (N.Forsgren et al., 2020).

In seeking alternatives to measure software delivery, CSE and SDP stand out as two models that offer clusters to classify software projects, together with a means of assessing the nature of software delivery and how it should be improved. The CSE software delivery measurement model (M.Barcellos, 2020) relies on an evolving pathway to maturity for its classification, particularly the adoption of continuous activities like CI, Continuous Deployment (CD), and Continuous Experimentation (CE). The CSE path to maturity consists of four ascending stages: **Agile Development**: i) the first maturity stage where teams, backlogs, and daily meetings are set; **CI**: ii) test-driven development, automated building, and a testing environment; **CD**: iii) agile sprints to deliver software, together with CI testing activities

to validate releases; iv) **CE**: experiments for a continuous assessment of customer data and feedback.

The SDP software delivery measurement model (N.Forsgren; J.Humble; G.Kim, 2018) offers a classification system that is based on achieved capabilities. This relies on four key metrics of DevOps: *Lead Time*: i) the time necessary for a change in software from its initial commitment to production; *Deployment Frequency*: ii) the frequency of released software; iii) *Mean Time to Restore (MTTR)*: the time required to restore software after an incident or bug; *Change Fail Percentage*: iv) the change failure rate. From the four key metrics of DevOps, SDP uses three clusters to divide the software projects into **High Performers**: those that performed above average; **Medium Performers**: those that had an average performance; **Low Performers**: those that performed below average.

Measuring software delivery through surveys has already been recommended by studies in large financial organizations (C.Vassallo et al., 2016), which rely on MSR to collect and analyze data (D.Costa et al., 2018). These make use of the SDP i.e. the four key metrics of DevOps (Lead Time, Deployment Frequency, MTTR, and Change Fail Percentage) and employ the following: available data sources (Logs and Issue Management system) (M.Sallin et al., 2021), interviews with practitioners (L.Lwakatare et al., 2019), and even mixed-methods consisting of a survey and a statistical analysis (E.Kula et al., 2018).

By following a similar pattern in the OSSP scenario, some studies have appeared that only cover the early stages of automation by suggesting ways to measure software delivery by means of MSR (B.Vasilescu et al., 2015; Y.Zhang et al., 2018; J.Bernardo; D.Costa; U.Kulesza, 2018), sample analysis (M.Hilton et al., 2016), or a manual analysis that entails looking for patterns (C.Vassallo et al., 2017). While an overall software delivery measurement is well covered, there is still a lack of studies on OSSP (S.Joshi; S.Chimalakonda, 2019), where the use, methods of implementation, and benefits of software delivery, can be explored in greater depth (M.Hilton et al., 2016). It can be concluded that there are still software delivery measuring opportunities to explore in OSSP.

Even though CSE and SDP models have been widely discussed in contemporary works in the literature (and these show there is an increased demand for knowledge about software delivery

measurement, as well as the commercial tools available - e.g. Oobeya, Swarmia, Linearb), there is still an opportunity to analyze the behavioral patterns of software delivery in popular OSSP. This is because of a lack of studies that cover its adoption, usage, implementation, and claimed advantages. Moreover, software delivery behavior is something dynamic, as the nature of decisions can, to a great extent, affect the current performance of software delivery performance (N.Forsgren; J.Humble; G.Kim, 2018). In light of this, a decision has been made in this study to include the following: a means for the automatic collection of software delivery metrics, that combine both CSE and SDP models, in popular OSSP with a view to analyzing and showing types of behavior, current classification outcomes, and trends. Given the dynamics of this subject, Releases are the main milestone to mark the collection of the metrics. MSR is used for data collection and, as its use is widespread, GitHub is the coding-hosted platform chosen.

1.2 Research Question

On the basis of motivational factors and the definition of the problem, this project investigates the following research question (RQ):

- **RQ**: Considering Software Delivery Performance as a software delivery measurement model, how do popular Open Source Software Projects behave?

1.3 Research Hypothesis

When seeking to answer the RQ raised, a combination of CSE and SDP models (in this study, simply referred to as SDP) can be used as a unified software delivery measurement model and, hence, will be able to clearly describe how popular OSSP behave. This has been already demonstrated in OSSP, where software delivery measurement is a reality and still provides new opportunities for research. Additionally, the research hypothesis also suggests that there is a demand for measuring and analyzing OSSP SDP behavior in an automated manner by taking account of the popular OSSP, as these projects are broadly adopted and evolving

software delivery is desired. Owing to the availability of APIs to collect data and the spread of OSSP in the source-code host chosen (GitHub), MSR will be the means of obtaining data by allowing a proper mapping from OSSP raw data inputs into SDP data. As soon as the SDP data have been preprocessed and collected, the process of getting metrics on a Release timeline basis is triggered by providing information and outcomes for further discussions and analysis.

1.4 Research Objectives

After obtaining OSSP data in an automated manner and in accordance with a combination of CSE and SDP software delivery measurement models, the main objective of this work is collecting SDP metrics from popular OSSP through a Release Timeline, and then analyzing the outcomes data. These particularly refer to the OSSP behavior concerning the languages of the collected projects, SDP Taxonomy both before and with the adoption of CI/CD, and an examination of the results data in a quantitative manner (through statistics and an exploratory study).

There are also some specific objectives (SO) for this study which are listed below:

(i) **SO1**: From a methodology that still has to be developed, map how popular OSSP behave on a Release timeline basis in terms of quantitative features: the total, median, minimum and maximum count of releases using the SDP Taxonomy both before CI/CD and with CI/CD, by means of three statistical tests: Shapiro-Wilk normality test, Wilcoxon signed rank test with continuity correction, and Cliff's Delta.

(ii) **SO2**: Publish the work through a replication package with the scripts and instructions needed for replication.

(iii) **SO3**: Conduct an exploratory study to evaluate the methodology developed with high-ranking popular OSSP.

1.5 Research Design

The research design defines a plan for how this study will answer the RQ, by describing each step as far as the final expected outcomes. Figure 1 shows the connection between the RQ, Research Hypothesis, the overall objective of this work, the research SO, the methodology, and expected outcomes. It starts from the RQ, (described earlier in Section 1.2), and then employs the research hypothesis defined in Section 1.3 to follow the overall objectives. Next, the overall objective is followed by the research SO, Section 1.4, that will underpin the methods required to achieve the expected outcomes.

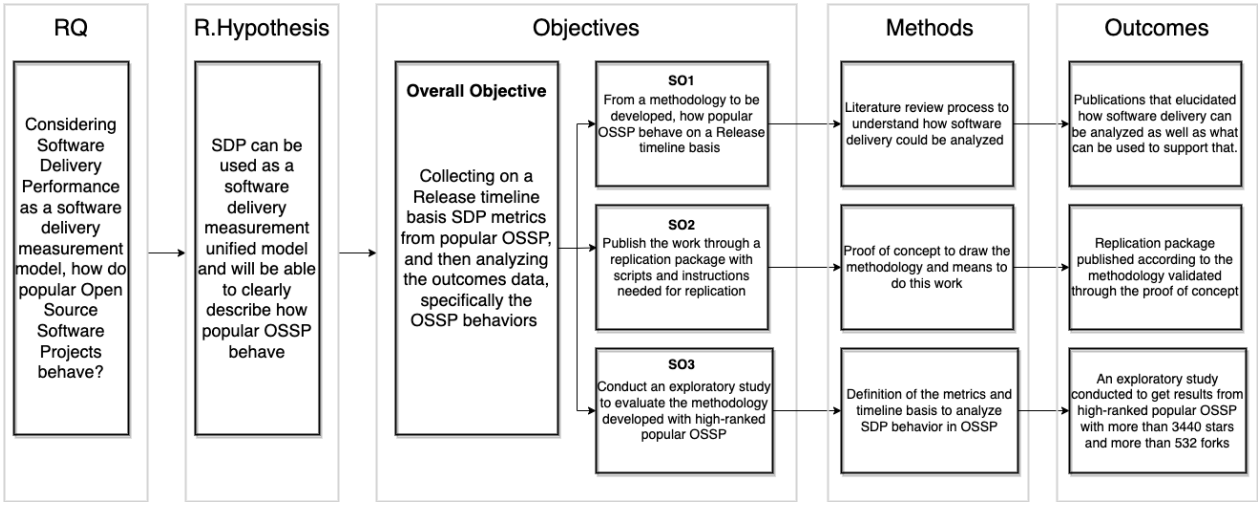

Figure 1: Research Design: Connection between the RQ, Research Hypothesis, the overall objective of this work, the research SO, methods, and expected outcomes

To answer the RQ (Considering Software Delivery Performance as a software delivery measurement model, how do popular Open Source Software Projects behave?), on the basis of the Research Hypothesis, this work aims to collect SDP metrics from popular OSSP on a Release timeline. Thus, three SOs were defined (SO1, SO2, and SO3) as explained in Section 1.4.

With regard to SO1 (From a methodology to be developed, how popular OSSP behave on a Release timeline basis), a literature review was carried out to understand how software delivery could be analyzed. This is described in greater detail, in Section 4.2.1. The outcomes consisted of publications that were designed to clarify how software delivery can be analyzed

and how it can enhance this study.

Concerning SO2 (Publish the work through a replication package with scripts and instructions needed for replication), the proof of concept is described in Section 4.2.2 confirmed the feasibility and potential of the work by enabling the development of a replication package that was in accordance with the final methodology. In respect of this SO outcome, the details of the replication package are described in Section 6.3.1.

Regarding SO3 (Conduct an exploratory study to evaluate the methodology developed with high-ranked popular OSSP), the definition of the metrics and timeline template to analyze the SDP behavior in OSSP, resulted in an exploratory study to obtain results from the high-ranking popular OSSP. The outcome of this exploratory study is also available with inputs and outputs (D.Barros; F.Horita; I.Wiese, 2023b).

1.6 Research Context

The Socio-technical Systems Research Initiative research group called Nexus Research Initiative [2] was officially founded at the Center for Mathematics, Computing and Cognition (CMCC) in UFABC. Nexus conducts research projects focusing on emerging technical summons (e.g., complex systems engineering, applied software systems) and the social concerns of communities and organizations (e.g., social and business transformation). In an effort to find solutions to its problems, Nexus possesses three branches of research: Complex System Engineering (**CSyE**), Applied Software Systems (**ASS**), and Society Business Transformation (**SBT**).

CSyE, the area of Nexus research that this study belongs to, is concerned with exploring, planning, and assessing socio-technical systems. Current research projects are devoting a great deal of effort to exploring the emergent behavior of system-of-systems, software delivery, and software modernization. **ASS** includes research projects concerned with developing and employing software to fix real-world problems. This research line contains works that are developing a Twitter visualization tool and geospatial data analysis software. **SBT** is planning research projects to understand and formulate theories on the role of systems in society and

[2] <https://pesquisa.ufabc.edu.br/nexus/>

business transformation. In this research area, there are studies that are examining the organizational constraints that arise from digital transformation. Figure 2 shows the Nexus Research Initiative foundations topics, along with their intersections and research goals.

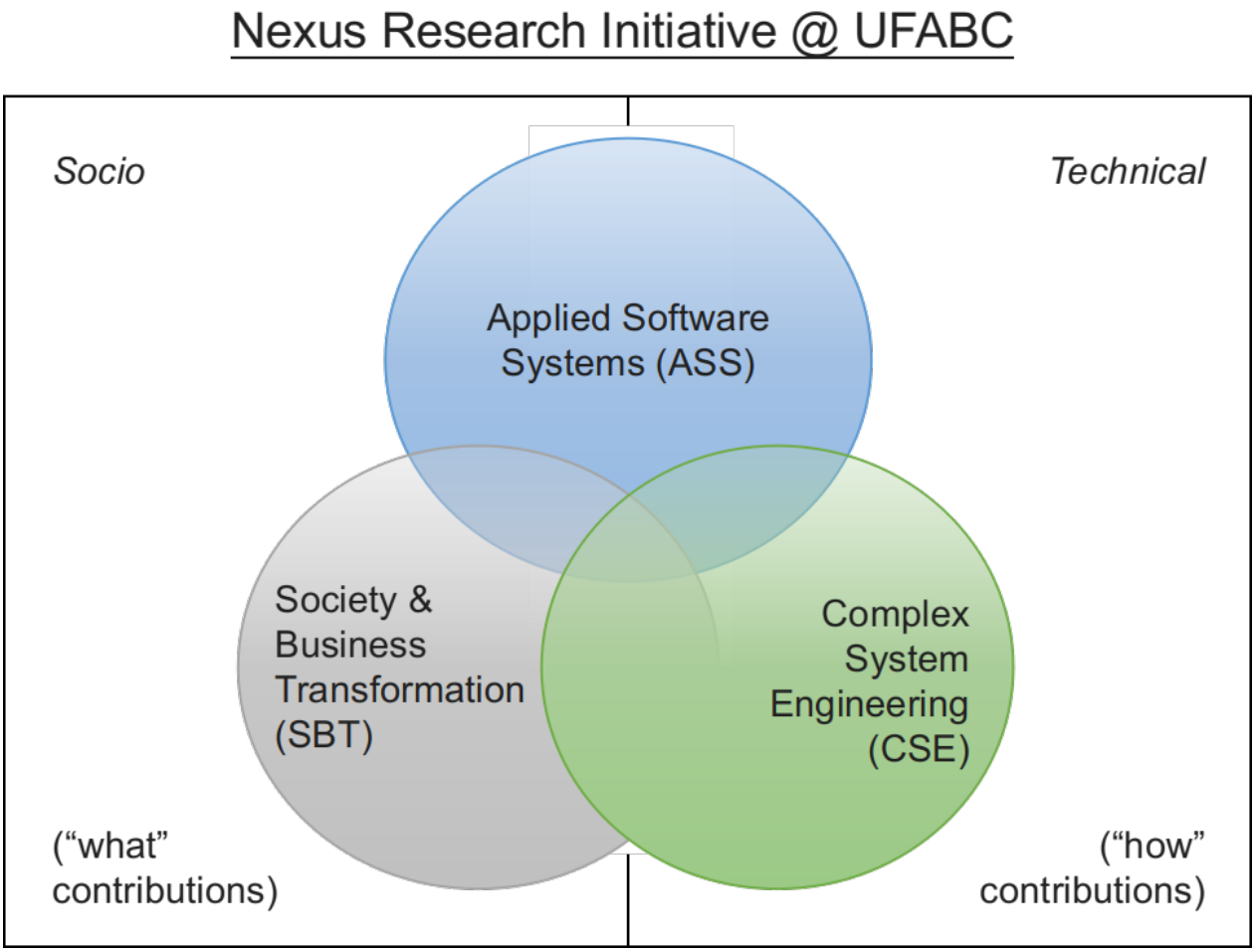

Figure 2: The Nexus Research Initiative foundations topics

The next paragraphs will briefly describe some ongoing and updated projects in the Nexus Research Initiative. There is one paragraph that describes the goals of each project, together with its academic and social benefits, and references.

A Modernization of Systems Methodology to Support Digital Transformation, based on a case study in the financial sector of Brazil. This research project (P.Leon; F.Horita, 2021) includes a modernized methodology that relies on digital transformation, or more specifically, the modernization of large-scale monolithic systems. It assumes that modern architectures can coexist with monolithic systems, and includes a case study of a

company in the financial sector in Brazil. The architecture and modernized system included allows for the gradual modernization of the system in line with the priorities of the business agenda.

Comprehending Emergent Behaviours in Systems-Of-Systems Through Software Simulation. Systems of Systems (SoS) are a number of large-scale complex systems, the components of which are independent systems themselves, and have their own features and functionalities. The interaction between the components of an SoS might lead to a new kind of behavior, known as emergent behavior. This research (K.Silva; F.Horita, 2021) seeks to collaborate with other researchers in broadening our understanding of this phenomenon, by means of modeling and simulation and showing how the inclusion and/or exclusion of one or more components of an SoS affects its global behavior.

Investigating the interplay of digital transformation and artificial intelligence in organizations. As digital platforms become more widely used in society, organizations are beginning to create business models to meet the needs of the new digital consumer. These changes are in response to the so-called Digital Transformation (DT) that characterizes the adoption of new technologies to modify the configuration of human resource management. At the same time, Artificial Intelligence (AI) provides more efficiency and an enhancement of customer experience. In this context, this study aims at exploring the relationship between DT and AI, as well as understand what might be the key linking elements. To achieve this., it has conducted interviews with digital leaders in organizations and analyzed the transcriptions through qualitative methods. The main findings show that the adoption of AI is often aligned with digital maturity (Veldhoven; J.Vanthienen, 2019).

1.7 Organization

This dissertation is divided into seven chapters, including this Introduction. The next Chapters are structured as follows. Chapter 2 sketches the Background. Chapter 3 describes the Related Works to this area of research. Chapter 4 carries out a literature review and shows how preliminary results can support the findings of this project. Chapter 5 discusses

the Methodology. Chapter 6 provides the research results and includes a discussion on the implications of the literature and practice on the RQ analysis. Finally, Chapter 7 summarizes the Conclusions and recommends future work on the basis of the research contributions, threats to validity, and opportunities for improving and developing the main topics investigated here.

CHAPTER 2

Background

2.1 Initial Considerations

This chapter is going to provide background related to important concepts this research will address and use. It is covering the two main approaches this work will combine to set a SDP model: (i) CSE and (ii) SDP.

2.2 CSE

The spread of software delivery is an established trend where companies are embracing it to deliver fast and assertive products to customers (S.Stavru, 2014). The number of success cases encourages companies to continue the quest for speeding software delivery (A.Sidky; J.Arthur; S.Bohner, 2007). However, the disconnection between continuous activities proposed by agile development methods could be more problematic rather than other practices (B.Fitzgerald; K.Stol, 2017). Therefore, the adoption of agile development methods alone is not enough, as to meet the customers and current market requirements there is the need for continuous planning, building, integrated operations, releases and evaluations (M.Barcellos, 2020).

Thus, to address the challenges of agile development methods implementation, especially the methods that propose continuous activities (e.g. CI, CD, CDE), CSE proposes an overall roadmap with a set of tools and practices to assist software development became a continuous,

quicker, interactive, integrated task aligned with business requirements (B.Fitzgerald; K.Stol, 2017). In place of a series of disconnected activities, CSE understands software development as a continuous movement, where software-related activities like planning, building, operation, and evaluation are performed in an interactive, flexible and collaborative manner.

As CSE is a new topic in Software Engineering, there are many opportunities being addressed (Karvonen et al., 2016) and questions to be answered (M.Barcellos, 2020). Moreover, there is no guidance yet on how CSE could be applied in a real-world case as the current literature still proposes manual and limited cases through interviews (H.Olsson; H.Alahyari; J.Bosch, 2012) or questionnaires with a small number of participants (Júnior; M.Barcellos; F.Ruy, 2021). For the purpose of formulating the whole picture of CSE, Barcellos (M.Barcellos, 2020) presented a CSE framework within a set of processes to be performed in their context and the relations between them.

Apart from providing an overview and knowledge concerning CSE, Barcellos' framework (M.Barcellos, 2020) took into account processes containing development activities performed simultaneously and gradually to lead companies into software maturity stages. Barcellos' work was inspired by related frameworks (H.Olsson; H.Alahyari; J.Bosch, 2012; B.Fitzgerald; K.Stol, 2017), wherein the typical evolution path of continuous delivery follows maturity stages with defined milestones. Although the definition of the maturity stages might be already known, this work is going to describe them in the next subsections under Barcellos' framework point of view, as they will be used herein methodology, results, and further discussions.

The maturity stages proposed by Barcellos are in this ascending order of software maturity. **Agile Development** is the first maturity stage proposed and it is featured by compact and organized teams, own backlog, short time cycles, and daily meetings. **CI** is the next maturity stage and it includes test-driven development with a dedicated environment, builds, and automation. As soon as the releases demand increases, the next maturity stage **CD** takes place. CD presumes frequent software deployment after agile sprints, as well as CI automation and tests. The last maturity stage is **CE**, where organizations allow continuous evaluation regarding implemented features through customers data and feedbacks (e.g., tests A/B).

Figure 3 illustrates the CSE framework proposed by Barcellos (M.Barcellos, 2020). The

maturity stages are defined by CSE core processes, represented by colored pentagons. There are also other processes, represented by rectangles with rounded edges, together with arrows indicating data flow from and to a repository database where information can be extracted for improvements.

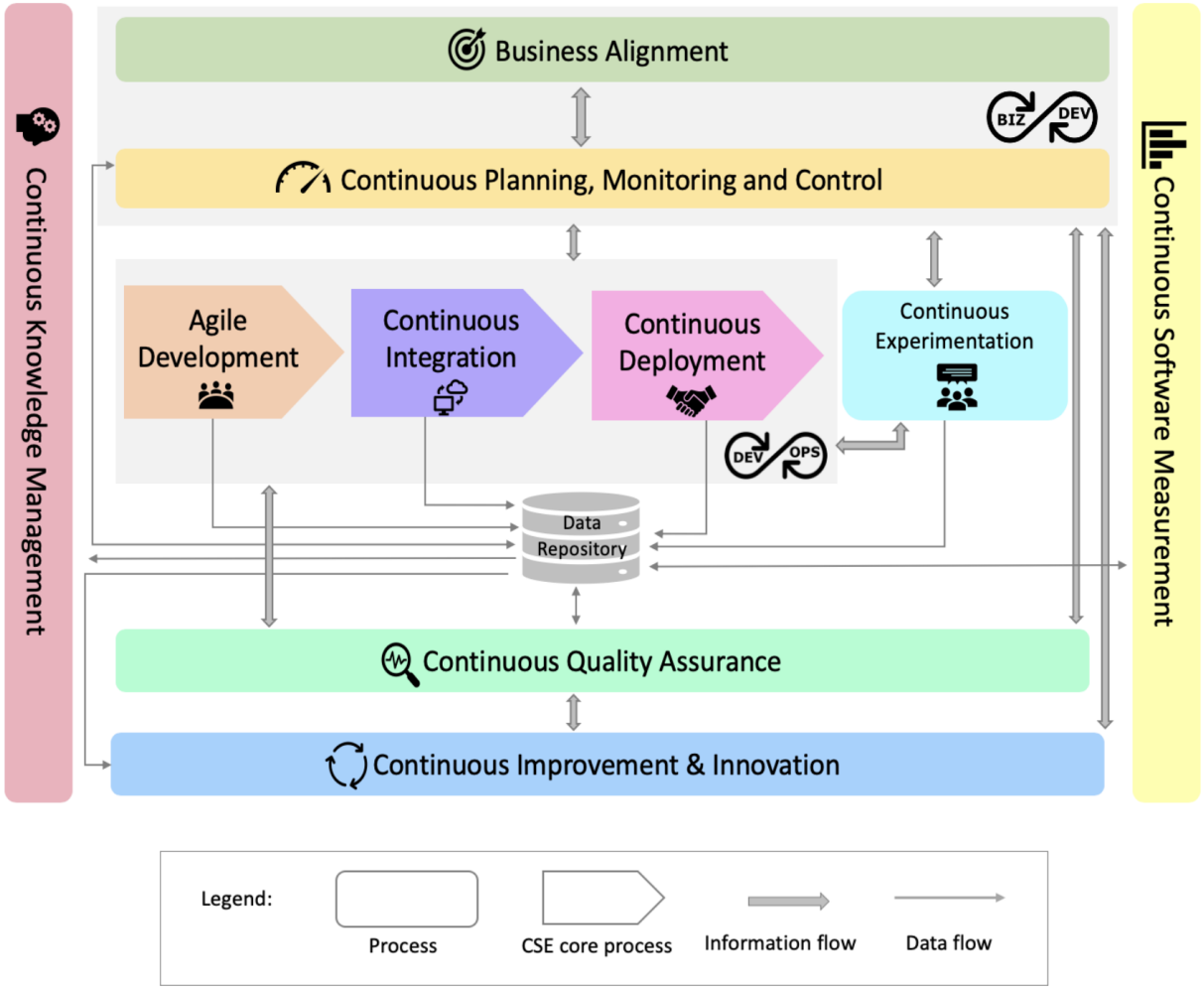

Figure 3: Framework for CSE proposed by Barcellos (M.Barcellos, 2020)

2.3 SDP

In the past, the Business as Usual (BAU) approach was the trend to deliver software projects (D.Kini, 2000). BAU projects used to have big timelines with long-time milestones. Currently, BAU approach is not enough to keep competitiveness, and companies are transforming how

products and services are delivered to customers: getting cohesive teams and providing fast software delivery cycles. As a means to understand and seek any improvements, this new way of delivering products and services to customers should have its performance measured through several vital capabilities such as technical solid practices, decoupled architectures, lean management practices, and trusting organizational culture (N.Forsgren et al., 2017).

Some assessments appeared to fulfill the gap in delivering product performance measurements like CMMI (M.Chrissis; M.Konrad; S.Shrum, 2011). However, most of them have relied on heavy and expensive interviews conducted by external consultants inside the firms, driving non-scalable and biased outcomes (N.Forsgren et al., 2017). Besides, as these assessments are conducted internally, so there is no external data, the benefits of outside benchmarking are not achieved either.

The main challenge is to accelerate software delivery processes, as most of the market interactions are through them. In this case, DevOps is suitable to support software delivery because it combines cultural changes and technology-enabled practices (A.Wiedemann et al., 2019). While several organizations have switched successfully the way of delivering products and services faster, there is still a lot of cases to be addressed, particularly the ones that need to increase DevOps adoption (N.Forsgren et al., 2020) . Furthermore, even the successful cases achieved in the past can demand new transformations and measurements.

To address the limitations of the assessments, the DORA platform (N.Forsgren et al., 2017) proposed a new way to perform assessment and benchmarking in three stages. The first one is building the assessment considering previous investigations of capabilities and driving improvements to deliver software. Secondly, there is an assessment refinement using psychometric methods that are valid and reliable, therefore coherent and replicable. Lastly, the DORA platform was built on a SaaS model allowing scalability and industry benchmarking. The main outcomes were initially drafted as IT performance four measurements: **Lead Time** for changes, **Deployment Frequency**, **MTTR**, and **Change Fail Percentage**. Figure 4 shows the DORA platform and its assessment tool process.

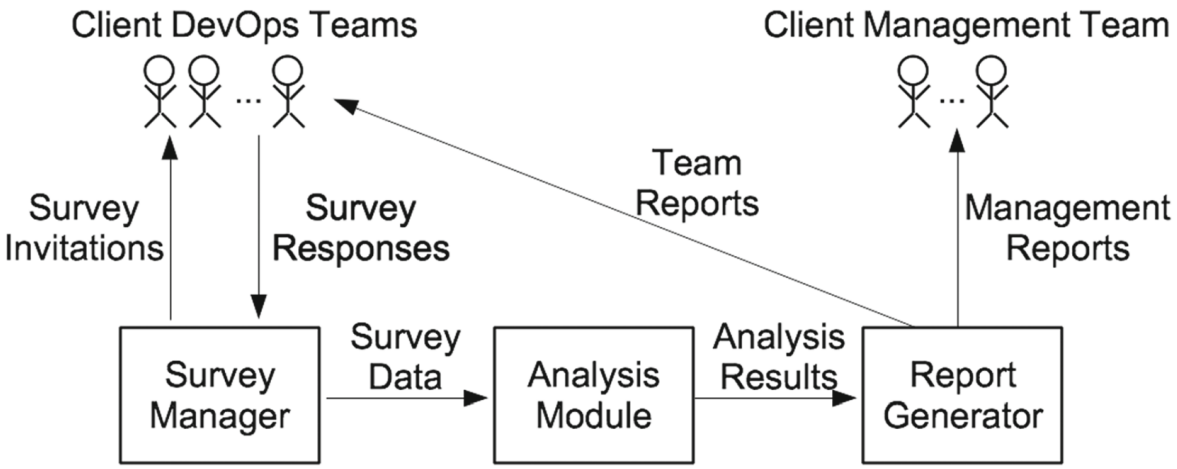

Figure 4: DORA platform proposed by Forsgren et al.(N.Forsgren et al., 2017)

Going forward to address the challenges to speed-up software delivery, Forsgren et al. (N.Forsgren; J.Humble; G.Kim, 2018) made a four-year research defined the SDP, collecting over 23000 surveys responses around the world in 2000 unique organizations, where the final output aims to provide capabilities and practices to accelerate software delivery in all types of organizations and projects. The research outcomes focused on IT delivery, therefore, the path from commit changes into software source-code to production (excluding other software development related processes, e.g. brainstorming, planning, requirements, or feasibility analysis).

The SDP research focuses on capabilities because, according to the authors, capabilities models attempt to help organizations continuously assuming dynamic and customized factors that might matter for each case. Capabilities models focus on outcomes and how they can be enhanced, instead of technical proficiency or tooling an organization posses. Even though Forsgren et al.(N.Forsgren; J.Humble; G.Kim, 2018) have uncovered 24 key capabilities driving overall enhancements in SDP, this work is focusing only on SDP CDE capabilities.

The mensuration of SDP CDE capabilities proposed a framework focused on outcomes settled on four metrics (N.Forsgren et al., 2020). **Lead Time** (for delivery) is the time it takes to go from a committed code to run in the production environment, in other words, this is commonly known as lead-time for changes. **Deployment Frequency** (or Release

Frequency) measures how often software code is released to production. **MTTR** is the time to restore service or mean time to recover. In other words, the MTTR metric calculates the average time it takes to restore software after an event that brought unavailability or impairs quality of service. **Change Fail Percentage** is the change failure rate, described as a percentage measure of how often deployment failures occur in production that requires immediate remedy (particularity, rollbacks).

Figure 5 shows the SDP framework proposed by Forsgren et al.(N.Forsgren; J.Humble; G.Kim, 2018). The authors used a data-driven technique called cluster analysis, where they grouped the surveys responses in a similar group aggregated. Each measurement contains its own dimension, and the range values from the statistical analysis fit the model to forward all responses to their group respectively. This cluster analysis was applied in all four years of Forsgren et al. research in the four measures of SDP (Lead Time, Deployment Frequency, MTTR, Change Fail Percentage), and the clusters proposed were *High Performers*, *Medium Performers* and *Low Performers*. With a focus on establishing a pattern and getting the latest clustering proposal, this research is using the last SDP cluster analysis and its parameters collected in 2017 as described in Figure 5, the last year of Forsgren et al. research.

	High Performers	Medium Performers	Low Performers
Deployment Frequency	On demand (multiple deploys per day)	Between once per week and once per month	Between once per week and once per month*
Lead Time for Changes	Less than one hour	Between one week and one month	Between one week and one month*
MTTR	Less than one hour	Less than one day	Between one day and one week
Change Failure Rate	0-15%	0-15%	31-45%

Figure 5: Framework for SDP proposed by Forsgren et al.(N.Forsgren; J.Humble; G.Kim, 2018) in the last year of the research

2.4 Final Considerations

This chapter provided the needed background this research is covering, specifically the two main approaches CSE and SDP. The next chapter will present the related work mapped, together with the opportunities this work is addressing.

CHAPTER 3

Related Works

3.1 Initial Considerations

This chapter will supply the main and most related works mapped so far relating to this work. It is covering a comparison between the contributions of the past works and the opportunities this work is addressing as well.

3.2 Related Works

The software delivery evolution is presented in the last decades in the literature. Whereas the adoption of agile development methods (like XP (Beck, 2000), BizDev (B.Fitzgerald; K.Stol, 2017)) is broadly used in software projects (P.Rodríguez et al., 2017), rapid release cycles have been attracting many adopters (M.Mäntylä et al., 2015). Apart from only adopting rapid releases, some works analyzed the outcomes of having smaller release cycles. From the adoption of rapid release in the Mozilla Firefox project, Khomh et al. (F.Khomh et al., 2012) provided a software quality study concluding that, although bugs are fixed faster, users experienced bugs earlier than expected. Still in the Mozilla Firefox, Mäntylä et al. (M.Mäntylä et al., 2015) looked over testing changes in software after the adoption of rapid releases and found rapid releases allowed a better investigation of the highest risk features.

Software delivery is well-covered in the literature (S.Joshi; S.Chimalakonda, 2019), and it

is possible to find surveys where CI is investigated in a large financial organization (C.Vassallo et al., 2016), as well as compiled a survey based on cases in ING (a Dutch global financial institution) to improve software delivery (E.Kula et al., 2021). Lwakatare et al. (L.Lwakatare et al., 2019) conducted interviews in five different companies to understand in practice how DevOps was implemented successfully. Kula et al. (E.Kula et al., 2019) conducted an exploratory study regarding rapid release at ING, noting benefits such as easier process to review code bringing more quality and drawbacks like dependency on infrastructure or testing. There are also works that attempted to compare both industry and OSSP projects, like a comparison of CI procedures and build failures through cluster analysis between Java OSSP and ING projects (C.Vassallo et al., 2017).

In OSSP there are already studies covering software delivery and public datasets that can facilitate new researchers, however, there is still an insufficient number of studies, opening opportunities for future works (S.Joshi; S.Chimalakonda, 2019). Additionally, OSSP studies have covered only the early stages of software delivery (Z.Khalil et al., 2021) like CI (Y.Yu et al., 2016) or Infrastructure-as-Code (Y.Jiang, 2016). Bernardo et al. (J.Bernardo; D.Costa; U.Kulesza, 2018) studied the impact of CI adoption in OSSP focusing on the investigation of Pull requests using MSR and Git local searches. Hilton et al. (M.Hilton et al., 2016) inspected several OSSP using GitHub API and then manually analyzed CI tools looking at the top 50 stargazed projects, double-checking the findings via survey.

Measuring software delivery is presented in the literature as well. There are disciplined approaches encouraging the adoption of agile development methods. Aiming to provide a structured process to guide organizations in adopting agile, Sidky et al. (A.Sidky; J.Arthur; S.Bohner, 2007) presented an agile adoption framework with an agile measurement indication with a four-stage process to achieve maturity. Following the same approach, the published work named as The Stairway to Heaven model (H.Olsson; H.Alahyari; J.Bosch, 2012) specifies an agile evolution path understood by five stages (Traditional development, Agile organizations, CI, CD, RD as an innovation system), where organizations should chase as a means to achieve maturity.

Concerning the CSE software delivery measurement model this work is using (M.Barcellos,

2020), there are conceptual frameworks as well as tools available in the literature. Karvonen et al. (Karvonen et al., 2016) proposed a holistic framework to map prerequisites for CSE adoption. A CSE diagnosis instrument built upon an electronic spreadsheet called Zeppelin (Júnior; M.Barcellos; F.Ruy, 2021) aims at evaluating the CSE promotion in organizations proposing artifacts to clarify a better view of the CSE current status and future enhancements. The Zeppelin usage was also tested to perform a survey with 28 Brazilian organizations aiming at investigating the adoption of CSE practices, suggesting that in specific situations organizations have a better adoption of agile and CD practices (R.Júnior et al., 2022). It is also possible to find options to collect CSE data partially like Garimpeiro tool (G.Destro; B.França, 2020), a web application that provides the level of CI/CD adoption using MSR to analyze local cloned GitHub OSSP. Although CSE has already been addressed in previous works, there is still an opportunity to collect automatically CSE data from popular OSSP in an organized and methodological manner, and thus combining CSE data with different models like SDP to be adaptable and answer new questions.

Regarding the SDP software delivery measurement model this work uses (N.Forsgren; J.Humble; G.Kim, 2018), there are published works addressing the SDP data collection like the four key metrics of DevOps (deployment frequency, lead time for change, time to restore service, and change failure rate) offered by Sallin et al. (M.Sallin et al., 2021), where the authors proposed an automated manner to collect data as well as a good evaluation of the value of measuring SDP DevOps metrics. There are also works supporting the SDP taxonomy, such as the investigation of the DevOps effects practices on the application of the Economic Order Quantity model (a predecessor to Lean Manufacturing) applied to software development (N.Forsgren et al., 2020). As an extension to measure SDP through developer productivity, some theoretical frameworks like SPACE (N.Forsgren et al., 2021), shown in Figure 6, propose different dimensions of productivity stating that these can be used to measure and improve software delivery. Here again, there is still an opportunity to collect automatically SDP data from popular OSSP in an organized a methodological manner from one single data source, as the authors of the previous works relied the collection on several disconnected data sources or ignore popularity. Besides, this work is filling the gap of combining two different models (in this case CSE and SDP) by proposing unprecedented methods and tools to do it in popular

OSSP through MSR.

FIGURE 1: **EXAMPLE METRICS**

LEVEL	SATISFACTION & WELL-BEING How fulfilled, happy, and healthy one is	PERFORMANCE An outcome of a process	ACTIVITY The count of actions or outputs	COMMUNICATION & COLLABORATION How people talk and work together	EFFICIENCY & FLOW Doing work with minimal delays or interruptions
INDIVIDUAL One person	*Developer satisfaction *Retention† *Satisfaction with code reviews assigned *Perception of code reviews	*Code review velocity	*Number of code reviews completed *Coding time *# Commits *Lines of code†	*Code review score (quality or thoughtfulness) *PR merge times *Quality of meetings† *Knowledge sharing, discoverability (quality of documentation)	*Code review timing *Productivity perception *Lack of interruptions
TEAM OR GROUP People that work together	*Developer satisfaction *Retention†	*Code review velocity *Story points shipped†	*# Story points completed†	*PR merge times *Quality of meetings† *Knowledge sharing or discoverability (quality of documentation)	*Code review timing *Handoffs
SYSTEM End-to-end work through a system (like a development pipeline)	*Satisfaction with engineering system (e.g., CI/CD pipeline)	*Code review velocity *Code review (acceptance rate) *Customer satisfaction *Reliability (uptime)	*Frequency of deployments	*Knowledge sharing, discoverability (quality of documentation)	*Code review timing *Velocity/flow through the system

Figure 6: The SPACE framework proposed by Forsgren et al.(N.Forsgren et al., 2021)

3.3 Final Considerations

This chapter presented the main and most relevant related works to this research. As stated here, there are opportunities in software delivery measurement, especially in popular OSSP, where the literature does not have studies covering all aspects described by models of software delivery measurement (i.e. CSE and SDP), nor proposing automated alternatives to get the needed data to analyze software delivery properly. In an effort to address some opportunities, this work is presenting a methodology where the CSE and SDP data will be mapped into the popular OSSP context by way of collecting data and getting metrics on a Release timeline basis. The next chapter will describe the literature review and preliminary results of this project.

CHAPTER 4

Literature review and Preliminary results

4.1 Initial Considerations

This chapter presents the literature review and preliminary results to support this project. Section 4.2 describes the steps to set the scope and goal, including a literature review, and preliminary results done through a proof of concept (POC) where part of this work could be validated. Finally, Section 4.3 provides final considerations.

4.2 Scope and Goal

The scope and goal of this methodology has been clarified and improved through the research, publications, proof of concept, and metrics selection done chronologically. Given the RQ proposed, the first step was to understand how software delivery could be analyzed and what could be the literature and tools to support that. So, a literature review process and publications described in Section 4.2.1 aimed to fulfill this gap. Secondly, after the reviews and publications, a proof of concept was necessary to verify whether SDP could be applied in OSSP, and this is detailed in Section 4.2.2.

4.2.1 Literature Review

The demand for measuring software delivery was the initial motivation of this work, and the RQ selected reflects it. In the process of reviewing the available literature, Forsgren et al. (N.Forsgren; J.Humble; G.Kim, 2018) proposed SDP as a well-cited and flexible compilation of best practices and capabilities categories that fitted the main goal of this work. Particularly, the CDE category was taken into account in this work motivated by the adoption of DevOps to measure performance. The relevance was confirmed by evolutions and works inspired by Forsgren et al. (N.Forsgren; J.Humble; G.Kim, 2018) like (N.Forsgren et al., 2020) and (M.Sallin et al., 2021).

From the understanding that projects could have their behaviors analyzed through SDP using CDE category with DevOps adoption, the next question was how it could be done, specifically for the OSSP scenario. For the purpose of doing so, the first related publication of this work (D.Barros; F.Horita; D.Fantinato, 2020) was a data mining tool called gthbmining.rc. The main goal was predicting DevOps trends, in this case, release candidates according with the architecture shown in Figure 7 where third-party libraries were adopted to get data from GitHub repositories and classify releases as candidates or not using the methods Naïve Bayes, Decision Tree (max depth=3) and K-Nearest Neighbors (number of neighbors=3).

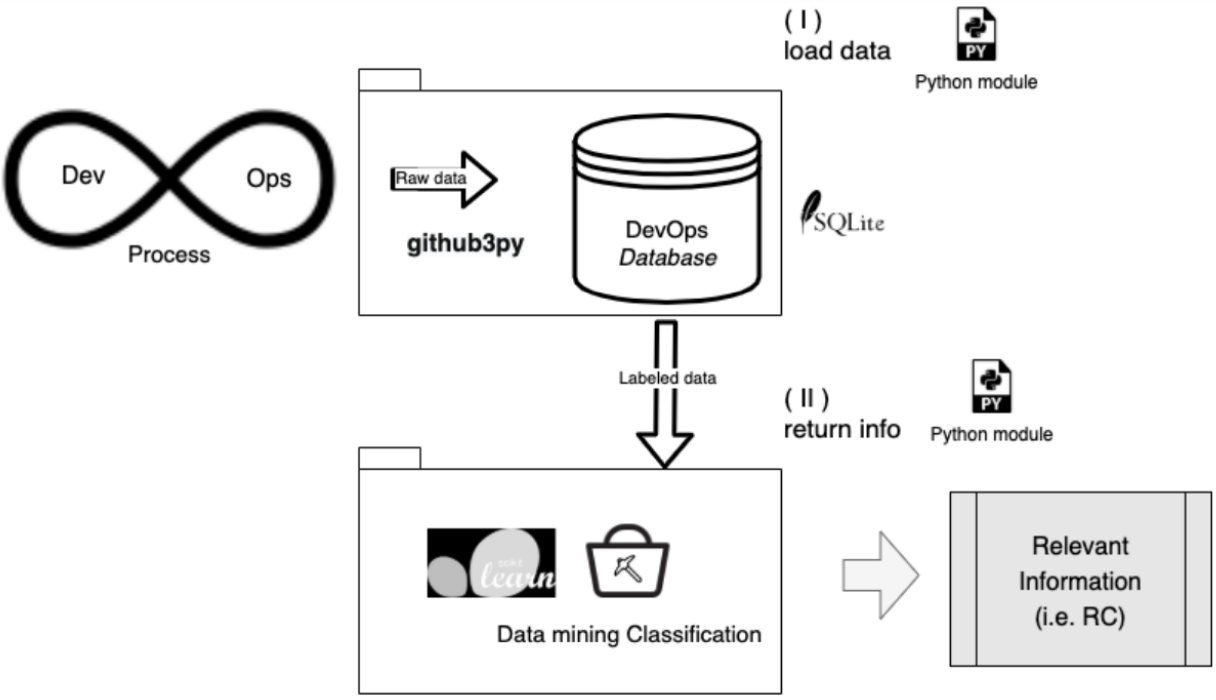

Figure 7: gthbmining.rc Architecture (D.Barros; F.Horita; D.Fantinato, 2020)

The next related publication of this work (D.Barros; F.Horita, 2020b) was a DevOps framework in a cyber-physical system case drawn in Figure 8, wherefrom releases raw data collected from DevOps cycle processes can be used to promote operational enhancements. Given the raw data stored, Data Mining and Machine Learning techniques were suggested to get relevant information, improving the Digital Transformation proposed.

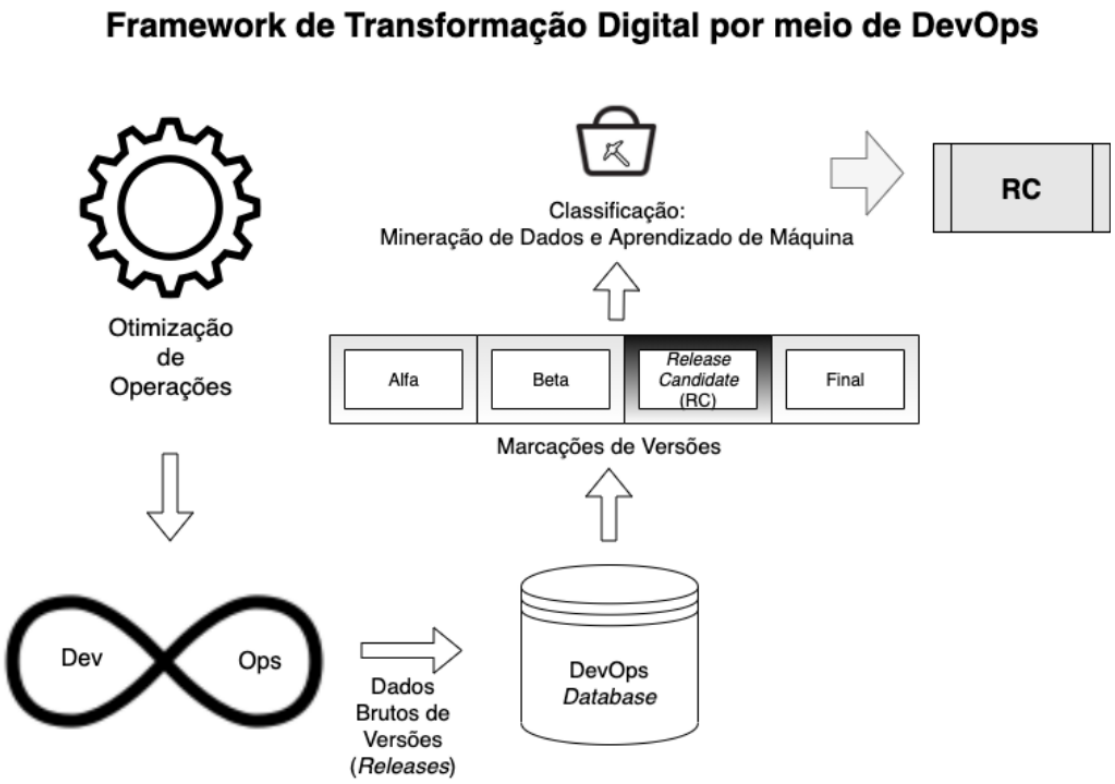

Figure 8: DevOps framework in a cyber-physical system case (D.Barros; F.Horita, 2020b)

There was also a publication (D.Barros; F.Horita, 2020a) where CDE was enhanced through Release Candidates (RC) supported by a Release Engineering Pipeline (REP). The proposed improvement is described in Figure 9 showing REP supporting CDE, and how it is improved through RC.

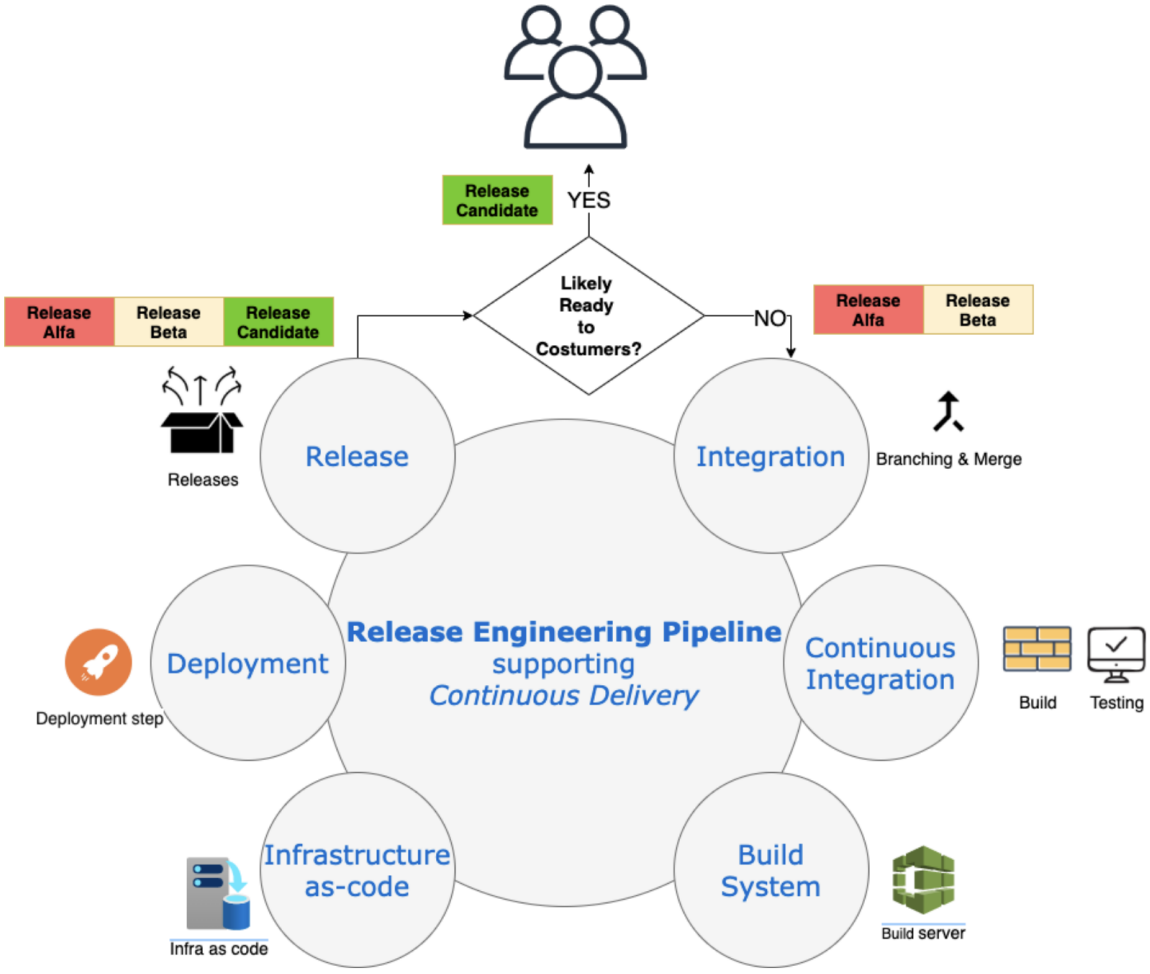

Figure 9: REP supporting CDE, and how it is improved through RC (D.Barros; F.Horita, 2020a)

Lastly, an MSR systematic mapping study to elicit and analyze the state-of-the-art was done (D.Barros et al., 2021), as MSR is one of the main tools used in this work. This study upgraded the MSR findings of the original MSR Cookbook (Hemmati et al., 2013) and then, via a systematic mapping study, proposed an extended version of the Cookbook keeping the same four high-level themes as Figure 10 shows. On top of that, the MSR extended cookbook contributed to this work by providing recommendations for all four high-level themes, suggesting the proper tooling, and stating lessons learned applied in one of the main MSR techniques used in this work.

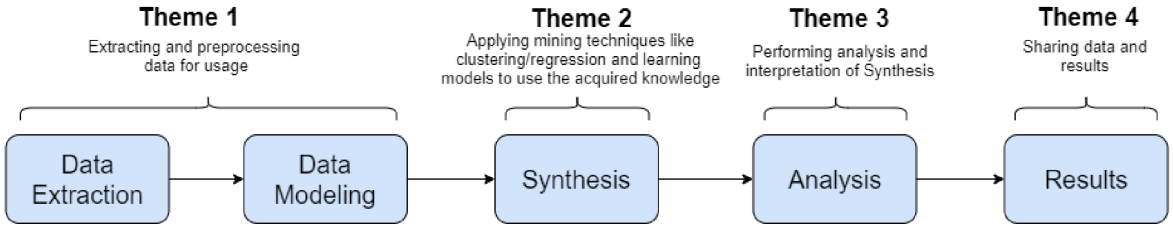

Figure 10: MSR Extended CookBook High-level Themes (D.Barros et al., 2021)

4.2.2 POC

Intending to draw the methodology and means to do this work, after the literature review, a POC was executed considering 1000 OSSP collected with the first version of this methodology, fully presented in Section 5.3, with database collected in June-2021 and available with all inputs and outputs (D.Barros; F.Horita; I.Wiese, 2023a). Out of the repositories collected, the POC also got 2470527 source-code files, 52119 Releases, 190201 Merged Pull requests, 263145 Closed Issues, 53657 Branches, 239035 Commits. Besides, from the 1000 OSSP collected 251 (25%) got all data needed and were considered for the Data Analysis.

In terms of OSSP development languages, the POC got the majority of TypeScripts projects (19%) followed by C++(16%), Java (12%), Python (12%), JavaScript (10%), Go (7%), C# and C(4%), Jupyter Notebook and Rust (3%), HTML (1%), and other languages (9%). Figure 11 draws the results grouped by language name.

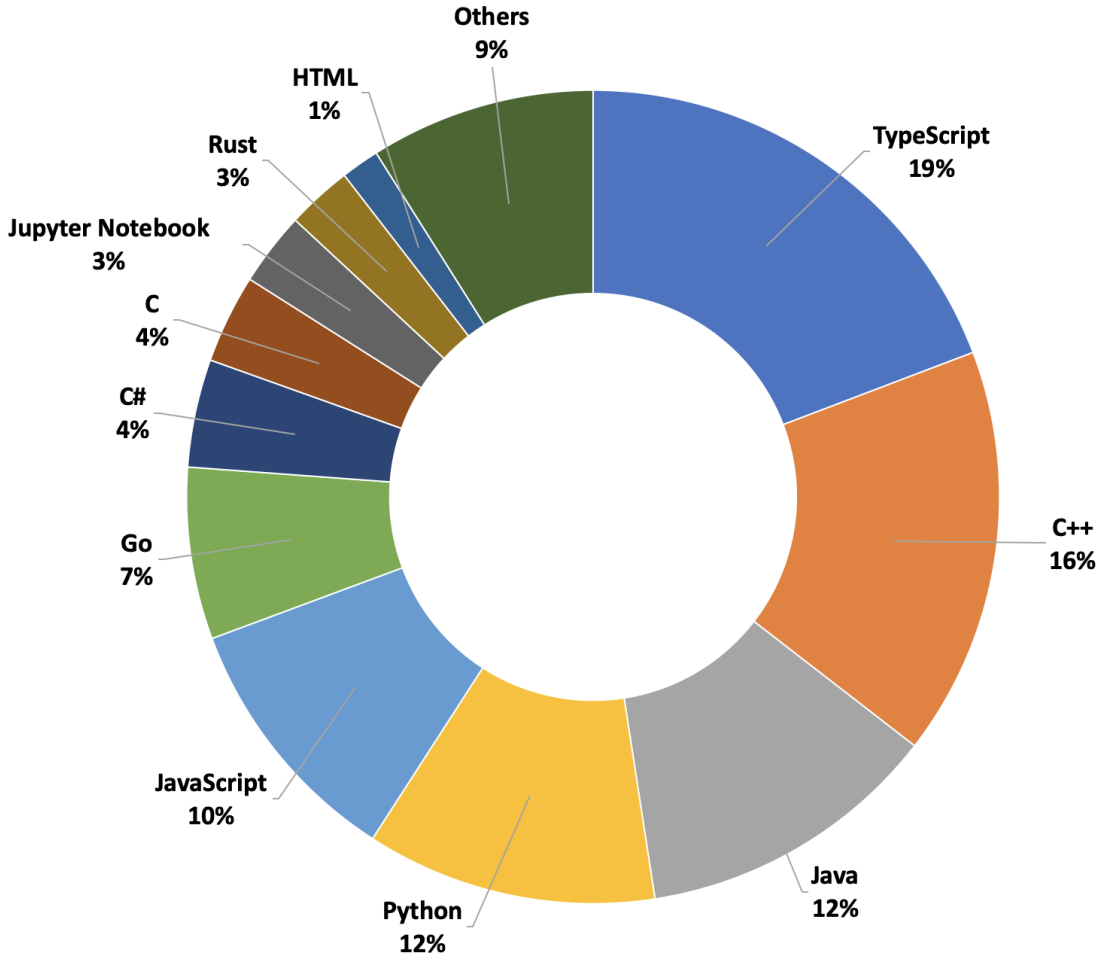

Figure 11: POC used Languages

As this was an exploratory study, there was no timeline set to calculate the metrics and classifications. The taxonomy classification process considered the metrics and values altogether setting always the worst classification scenario. For instance, if one project got a High Performer classification for three metrics but a Low Performer for one, it would be set to Low Performer. SDP Taxonomy got values for Low Performers and Medium Performers, leaving High Performers outside. Table 1 exhibits the Data Analysis for SDP Taxonomy classifying process in the POC.

Table 1: Data Extraction classifying data in the POC

SDP Taxonomy (N.Forsgren; J.Humble; G.Kim, 2018)		
Cluster	***Quantity***	***Percentage***
Low Performers	236	94%
Medium Performers	15	6%
High Performers	0	0%

Inspecting the metrics calculated together as a group, it is possible to see a majority of values as Low Performers (motivated mainly due to MTTR values) and the rest as Medium Performers for the SDP Taxonomy based on capabilities. To clarify this quantitative analysis, the calculated metrics from SDP Taxonomy are plotted using Box Plot diagrams to see the variability of data. Figure 12 presents the Box Plots for all SDP metrics (Lead Time, Deployment frequency, MTTR, Change Fail Percentage, all properly explained in Section 2.3). It is also possible to see that metrics analyzed individually got different scenarios, such as the median for MTTR classified as Low Performer against the median for Change Fail Percentage classified as High Performer.

Figure 12: POC SDP Metrics Box Plot

Regarding the popularity of OSSP, it is possible to see that the given star and requested forks values increase the capabilities classification. Table 2 shows the SDP Taxonomy

popularity data in the POC.

Table 2: Data Extraction popularity data in the POC

SDP Taxonomy (N.Forsgren; J.Humble; G.Kim, 2018)						
	Given Stars			***Requested Forks***		
Cluster	***Avg***	***Min***	***Max***	***Avg***	***Min***	***Max***
Low Performers	29949	15286	184658	5468	598	84949
Medium Performers	38891	16229	117441	7506	988	28569
High Performers	0	0	0	0	0	0

It is possible to see graphically the OSSP SDP Taxonomy trend reported in Table 2. Apart from the maximum values, the average and minimum values of given stars and requested forks follow an increasing trend when comparing the evolution of performers. Figure 13 draws the numbers and trend lines for Low Performers and Medium Performers, the two clusters found for SDP Taxonomy in results. The green arrows indicate an uptrend, and the red arrows indicate a downtrend.

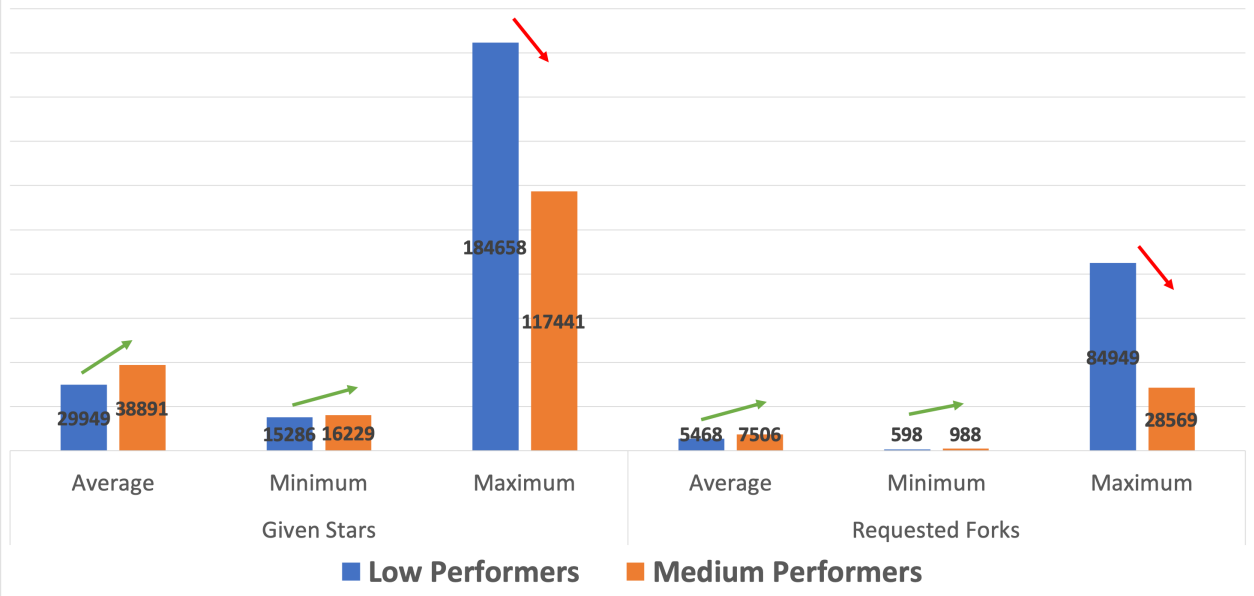

Figure 13: POC OSSP SDP Taxonomy trend

Apart from the SDP Taxonomy, the POC also calculated values for the CSE Framework to understand if a path to maturity, considering the adoption and evolution of continuous

activities like CI or CD, was something to be included in this work. Table 3 shows the CSE classification with a preponderance of Agile Development (79%) against CI (21%). There were no OSSP classified as CD nor CE.

Table 3: Data Extraction classifying data in the POC

CSE Framework (M.Barcellos, 2020)		
Cluster	***Quantity***	***Percentage***
Agile Development	199	79%
Continuous Integration (CI)	52	21%
Continuous Deployment(CD)	0	0%
Continuous Experimentation (CE)	0	0%

Additionally, regarding the popularity of OSSP, it is possible to see that the given star and requested forks values increase when the OSSP CSE Framework maturity increase as well. Table 4 shows the CSE Framework popularity data in the POC.

Table 4: Data Extraction popularity data in the POC

CSE Framework (M.Barcellos, 2020)						
	Given Stars			***Requested Forks***		
Cluster	***Avg***	***Min***	***Max***	***Avg***	***Min***	***Max***
Agile Development	28924	15286	170115	4851	598	73688
Continuous Integration (CI)	36452	15408	184658	8419	1178	84949
Continuous Deployment(CD)	0	0	0	0	0	0
Continuous Experimentation (CE)	0	0	0	0	0	0

Figure 14 shows graphically the OSSP CSE Framework trend reported in Table 4. Drawing the values of Agile Development and CI, the two clusters found for CSE Framework in results, it is possible to see, through the green arrows, the evolution of given stars and requested forks average, minimum, and maximum values across the enhancement of the level of maturity.

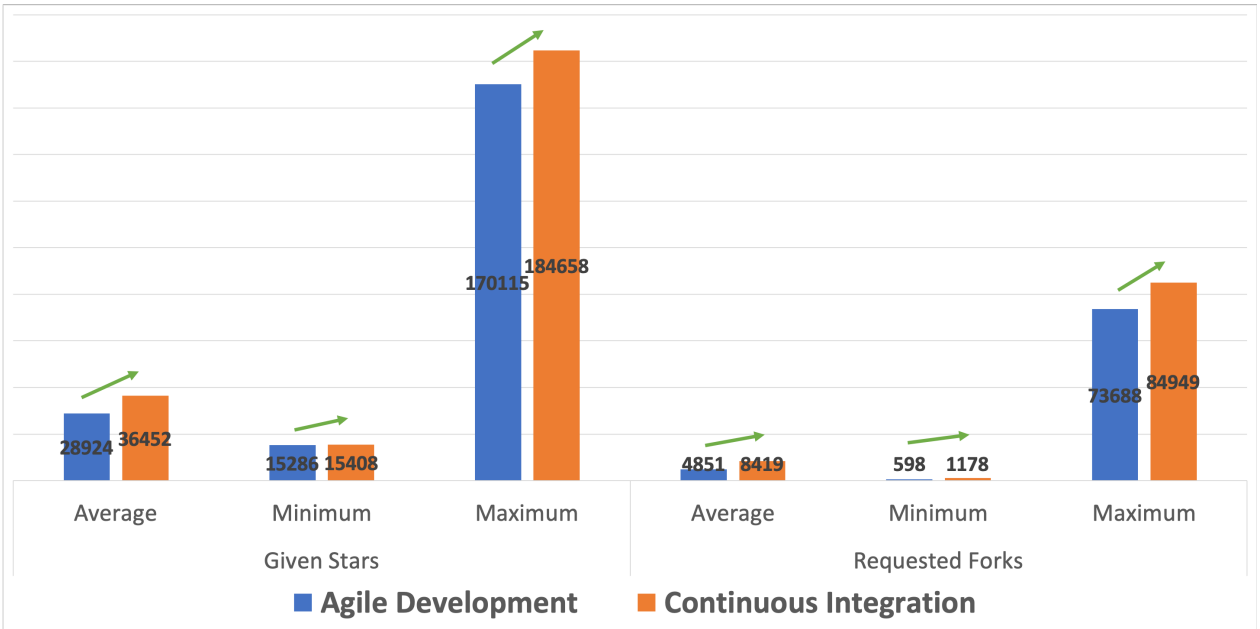

Figure 14: POC OSSP CSE Framework trend

4.3 Final Considerations

This chapter presented the literature review and preliminary results for this project, elucidating the scope and goal, as well as bringing important outcomes do define the methodology. The next chapter will describe this work methodology.

CHAPTER 5

Methodology

5.1 Initial Considerations

This chapter presents the methodology for this project. Section 5.2 defines the metrics chosen based on the previous outcomes and reviews. Section 5.3 defines the methodology aggregating Section 5.3.1, that explains the data extraction together with its structure and load data process definitions from the defined goal, and Section 5.3.2, that elucidates the data analysis focusing on how metrics are calculated and how final results are generated and become available for further investigation. Finally, Section 5.4 provides final considerations.

5.2 Metrics Chosen

The literature review and the POC described in Chapter 4 were essential to justify the initial assumptions this work needed to confirm and select the metrics to analyze SDP behaviors in OSSP, as this is the main goal of this work.

On one hand we have the literature review and publications to state that SDP is an efficient way to measure projects performance, and that DevOps variables can be found using MSR. On the other hand, the POC clarified that SDP using CDE category and its taxonomy can be applied in OSSP scenario.

The POC results confirmed popularity, represented by the OSSP number of stars and requested forks, is an initial filter to be considered. Figures 13 and 14 validated graphically that increasing popularity improves the OSSP classifications, and this is true looking into either SDP Taxonomy or CSE Framework. Therefore, the number of stars and requested forks shall be defined as input parameters to filter the most popular OSSP.

While the POC considered only the current situation, a definition of a timeline basis was needed to calculate the metrics and classifications to see the OSSP improvements thought for a specific time. Metrics should be calculated considering software CDE, so Releases are the best milestones to act as a timeline basis as they represent a means of boxing and shipping software to users.

By way of MSR, the POC metrics results proved the relevant ones to be chosen in this methodology. Looking at the outputs the CSE Framework provided, the CI/CD adoption is one relevant metric ratified by the feasibility to collect it as well as the results of the evolution of maturity motivated by popularity shown in Figure 14. This evolution demonstrated that, in fact, continuous activities improve the projects' productivity and evolutions as quoted in others works in literature too (B.Vasilescu et al., 2015). Considering the outcomes explained here, CI/CD adoption is one of the metrics to be included in this methodology.

Still in the metric selection and regarding the SDP Taxonomy, apart from the feasibility to collect all metrics, the values shown in Figure 12 indicated that some metrics could change substantially the final results trending them to above the reality, as Change Fail percentage did, or below the reality, as MTTR did. Besides, the two metrics Change Fail percentage and MTTR rely on weak factors that may affect negatively the analysis of the final results: Change Fail percentage uses OSSP branches to be calculated and, if there is no knowledge to deal properly with branches, branching can drive opposite effects towards performance (Store, 2020); MTTR uses Issues to be calculated and this can also mislead any analysis, as Issues can be reported in many ways across OSSP (T.Bissyandé et al., 2013), predicting its types would require additional tooling and efforts unforeseen in this work (R.Kallis et al., 2021), and the time to address an issue can be not enough to measure how long it will be available in the software (D.Costa et al., 2014). Hence, the SDP Taxonomy metrics chosen in this

methodology are Lead Time and Deployment Frequency, leaving out Change Fail percentage and MTTR.

5.3 Definition

The definition of the methodology this work uses took into consideration the literature review, the POC results, and the chosen metrics, all described in the previous section. The main goal here is to analyze the OSSP behaviors through SDP using CDE category with DevOps adoption and the literature review and publications support this assumption. The POC gave confidence that popularity should be one of the main filterings, and OSSP data can be collected via MSR in an efficient and accurate manner. Furthermore, the metrics chosen by the POC results relied on calculated results, so a timeline basis appeared as a need to compare outcomes grouped by Releases.

Figure 15 draws the methodology used in this work, where there are four numbered milestones (from one to four) grouped by two modules: **Load Data**(further down described in Section 5.3.1): where all OSSP data is collected from the host platform (GitHub), appropriately stored in a local relational database, and pre-processed to allow the process of collecting metrics and generating final results. This component is responsible for milestones *1* and *2*; **Returning Information** (further down described in Section 5.3.2): wherefrom the process of collecting data in the **Load Data** step, SDP data are classified according to their taxonomy defined in Section 2.3, using the metrics chosen (*Lead Time*, *Deployment frequency*, *CI/CD adoption*) on a Release timeline basis. Then, the final results are available for further analysis and comparison. This component is responsible for milestones *3* and *4*; The source code of this work is available at GitHub as well in a replication package[1] to allow reproducibility and motivate future researchers and advancements.

[1] <https://github.com/ddangelorb/gthbmining/tree/master/sdp>

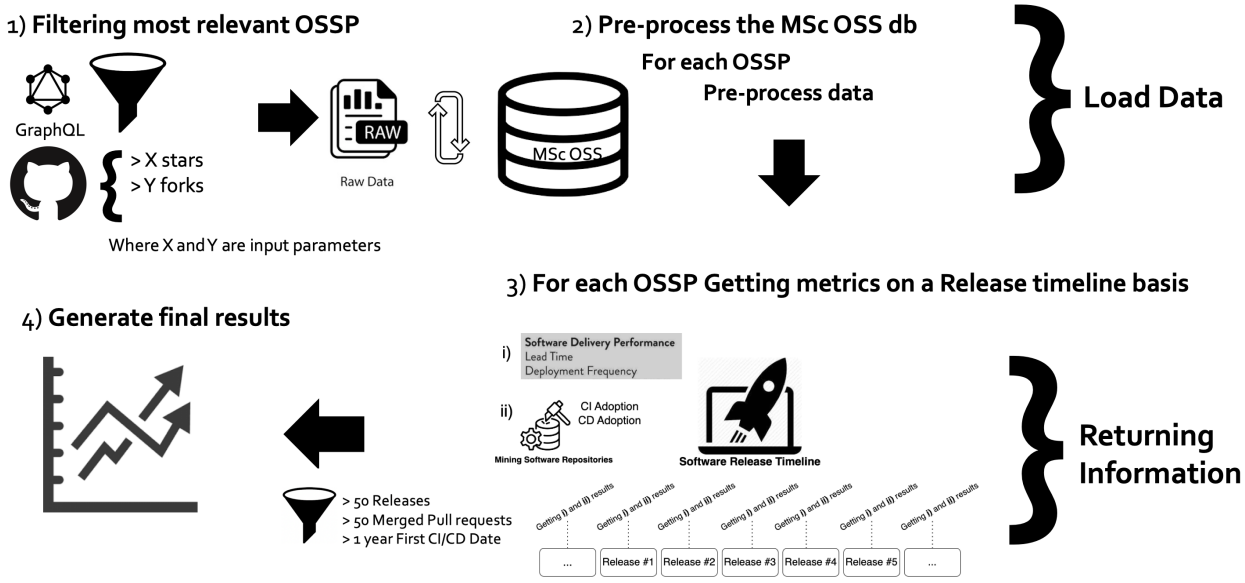

Figure 15: Methodology proposed

5.3.1 Data Extraction

The Data Extraction is composed of the *Data Structure* this work uses, together with the data collecting process called here *Load Data*. These two are detailed below.

Data Structure GitHub[2] is the only data source used and, apart from it, there is no more additional database nor system to collect data. GitHub is one of the most important software development platforms, providing hosting for software development version control and collaboration using Git as a version control tool. According to statistical data collected in August 2022 from the GitHub Octoverse website (GitHub, 2022), GitHub possesses more than 73 million developers, 61 million new repositories created in one year, and 170 milion of merged Pull requests.

GitHub organizes projects using Repositories, called here in this work as OSSP. The statements to collect and store information follow the data structure GitHub possesses for all OSSP hosted: each OSSP can have several Releases (versions ready to be deployed and available for usage) with individual dates; OSSP Pull requests (source-code changes proposed

[2] <http://github.com/>

individually by a user to fix an Issue or implement something new), grouped by dates, can possess several Commits (changes to a file or a set of files) along with their status (i.e. open, closed, merged); OSSP files contain their paths, and their history of versions. Figure 16 shows the GitHub data structure, as well as its main entities relationship relating to OSSP this work assumes.

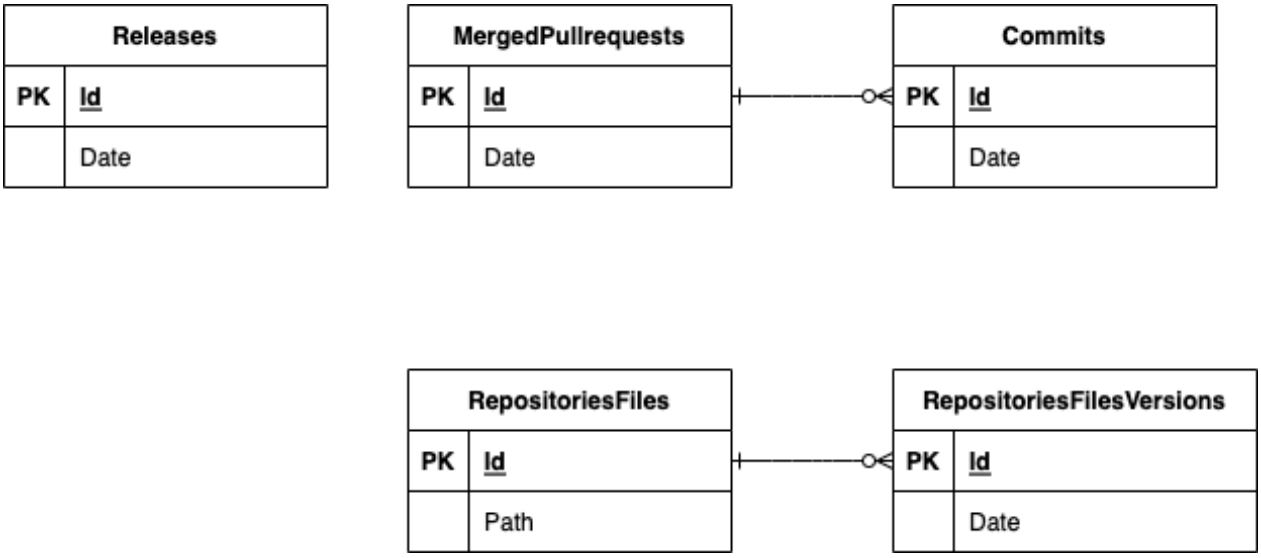

Figure 16: OSSP GitHub data structure

Load Data The Load Data component contains the two milestones (1 and 2) described next.

1.Filtering most popular OSSP: This milestone starts getting raw data from the most popular OSSP. Intending to select the best filtering to get the most popular OSSP, this work utilizes an approach where the project's popularity is taken into account (H.Borges; A.Hora; M.Valente, 2016). In this way, two variables are available as input parameters to set a minimum amount of stars (variable X) and a minimum amount of forks (variable Y). Therefore, OSSP with more than X stars and more than Y forks will be included regardless of other attributes like language, number of Releases, Commits, and Pull requests. For each OSSP selected, by way of the GitHub GraphQL API v4 [3], the raw data is sent to the next milestone, where data are prepared for getting metrics and generating results.

[3] <https://developer.github.com/v4/>

2.Pre-process the MSc OSS db: This milestone starts the preprocessing step, where the raw data received is properly stored into a local database called **MSc OSSP** normalizing data for the Returning Information component. In other words, here in the milestone 2, data is handled to make ready the database to the next component classify and correlate CSE and SDP data. The **MSc OSSP** database, described in Figure 15, holds all entities related to each OSSP collected, represented centrally through the entity *Repositories*.

From the entity *Repositories* the collected data are stored in other entities and related ones needed for SDP data calculation: *RepositoriesFiles* (details of all files in the repository), *Releases* (all releases the repository owns), *MergedPullrequests* (repository pull requests that were merged), *Commits* (repository commits linked to a pull request), *RepositoriesBranches* (repository available branches), *RepositoriesLanguages* (repository languages proportionally by its software development). There are also entities and related ones to support and store SDP data calculations: ClassificationMetrics (holds repository sdp metrics, i.e. Lead Time, Deployment frequency, CI usage, CD usage), *ClassificationsMetricsReleases* (links each ClassificationMetrics value to a Release, setting the values accordingly), *SDPClassifications* (holds repository sdp classification, i.e. High Performers, Medium Performers, Low Performers), *SDPClassificationsReleases* (links each SDPClassifications value to a Release, setting the values accordingly). Figure 17 shows the entity-relationship diagram this work uses within the entities described.

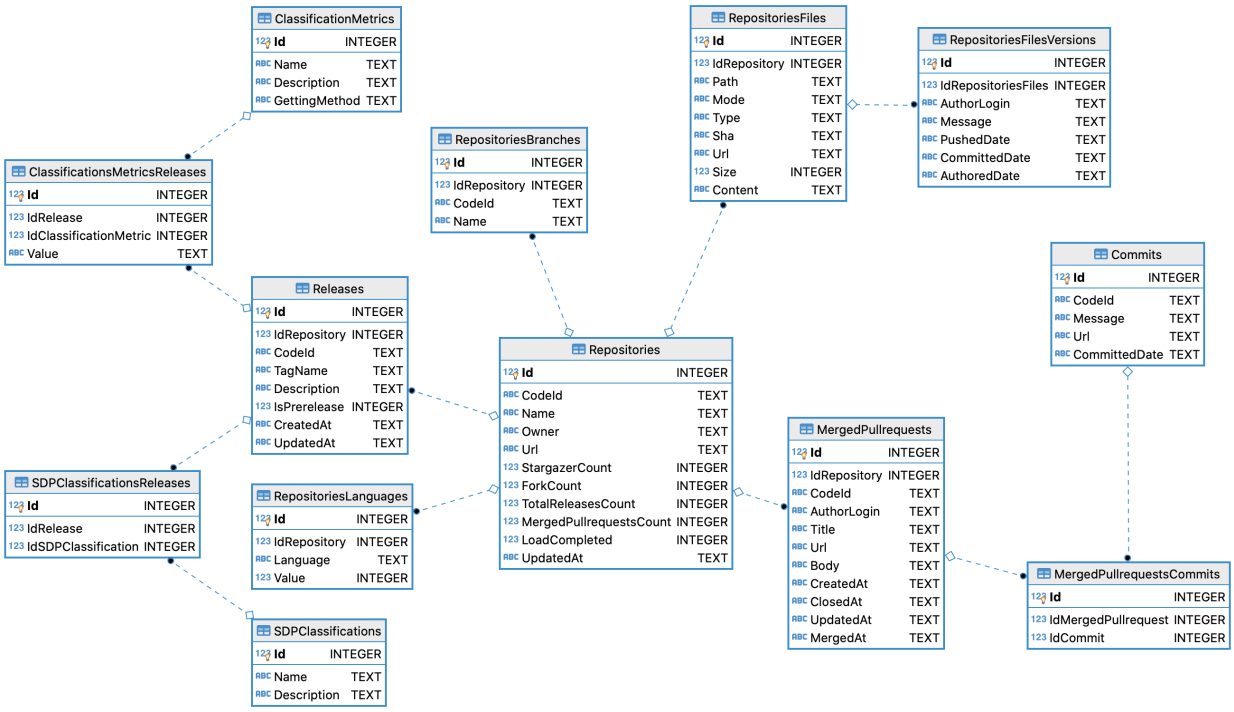

Figure 17: Methodology Entity Relationship Diagram

5.3.2 Data Analysis

The Data Analysis can be executed after the Data Extraction. It consists of the information generating process called here *Returning Information* detailed below.

Returning Information The Returning Information component contains the two milestones (3 and 4) described next.

3.For each OSSP Getting metrics on a Release timeline basis: Here, the component selects only OSSP that got all data needed for getting metrics filtering only OSSP that have at least: total count of fifty Releases (represented by *Releases* in Figure 17), total count of fifty merged Pull requests (represented by *MergedPullRequests* in Figure 17). Moreover, an additional cleaning process is set to consider only OSSP with more than one year from the extraction date with a CI or CD tooling adoption.

For each OSSP selected considering the filtering described in the last paragraph, the

getting metrics process starts on a Release timeline basis. It means that for each OSSP all Releases are selected, and for each Release the metrics *Lead Time*, *Deployment Frequency*, *CI adoption*, and *CD adoption* are calculated. There are metrics statements that should be considered in the calculation process described next.

SDP classifying statements (applied for the metrics *Lead Time* and *Deployment frequency*) From the SDP Background presented in Section 2.3 (where the SDP definitions and metrics were explained) together with the GitHub data structure described in Subsection 5.3.1, this work is proposing a mapping of the two metrics chosen from the CDE capabilities by Forsgren et al. (N.Forsgren; J.Humble; G.Kim, 2018) (*Lead Time*, *Deployment frequency*) into OSSP context using MSR. For each CDE capability, data will be collected through the available APIs and will be examined as the following statements.

- **Lead Time**: It is obtained considering the commit time until production. In the GitHub OSSP scenario, Releases are considered software versions ready to go live. Pull requests are considered the main way to propose source-code changes, and the status merged suggests it has been accepted. Besides, there is a link between Pull requests and Commits (concepts and relationships described properly in the Subsection 5.3.1).

  From the process of collecting OSSP Pull requests, Commits, and Releases, represented by the entities *MergedPullrequests*, *Commits*, and *Releases* (described in the Subsection 5.3.1), for each project, this metric is calculated getting the average time in hours as follows: i) For each Pull request get the latest Commit; ii) Associate the Commit to the closest Release considering Commit date against the Release date.

- **Deployment frequency**: It is the Release frequency. In the GitHub OSSP scenario, as described before, Releases are considered software versions ready to go live.

  From the process of collecting OSSP Releases represented by the entity *Releases* (described in the Subsection 5.3.1), for each project, this metric is calculated getting the average time in hours considering all the Releases dates.

As soon as the CDE metrics are calculated, the OSSP analyzed are classified in the database

following the taxonomy proposed by Forsgren et al. (N.Forsgren; J.Humble; G.Kim, 2018): **High Performers**: those who got metrics results above average; **Medium Performers**: those who got metrics results on average; **Low Performers**: those who got metrics results lower on average. Following the same logic adopted in the POC, the taxonomy classification process considered the metrics and values altogether setting always the worst classification scenario. For instance, if one project got a High Performer classification for one metric but a Low Performer for another one, it would be set to Low Performer.

For a matter of choice, as discussed in Section 2.3, this project uses the last SDP taxonomy proposed by Forsgren et al. (N.Forsgren; J.Humble; G.Kim, 2018) dated from 2017 indicated in Figure 5. Consequently, **High Performers** are OSSP where *Deployment frequency* happens multiple times per day, Lead Time less than one hour; **Medium Performers** are OSSP where *Deployment frequency* happens between once per week and once per month, Lead Time between one week and one month; **Low Performers** are OSSP where *Deployment frequency* happens between once per week and once per month, Lead Time between one week and one month.

CSE classifying statements (applied for the metric *CI/CD adoption*) Considering the CSE Background shown at Section 2.2, this work is proposing a mapping from the maturity stages proposed by Barcellos (M.Barcellos, 2020), particularly CI and CD into OSSP context using MSR. The main goal here is to identify whether each OSSP adopts a CI/CD tool setting it as a metric.

- **CI Tool**: CSE professionals frequently select tooling approaches to define and support CSE (J.Johanssen et al., 2018), so the CI tools mapping is relevant in this maturity stage. This work initially considered the CI tools based on a systematic review, where CI state of the art was reviewed classifying approaches and tools through stages *Build System* and *CI Server* in a suggested deployment pipeline (M.Shahin; M.Babar; L.Zhu, 2017). However, as the systematic review considered known best practices, it was necessary to adapt the output tools into the OSSP scenario. Thus, Build and CI tools that could not be fit in the OSSP context completely were eliminated from the systematic review

study: Bamboo, Hudson (discontinued tool), MS Build, NAnt, Tinderbox (discontinued tool). Likewise, to describe better OSSP context some CI tools were added: Buddy, CircleCI, TravisCI (S.Mohammad, 2016), Gradle (T.Rausch et al., 2017). Hence, the final CI tools this work contemplates are: Ant [4], Buddy [5], CircleCI [6], CruiseControl [7], Gradle [8], Hydra [9], Jenkins [10], Make [11], Maven [12], Sysphus [13], TeamCity [14], TravisCI [15], GitHub Workflows [16].

From the process of collecting OSSP files represented by the entity *RepositoriesFiles* (described in the Subsection 5.3.1), for each project, there will be a search in each file path looking for patterns to prove the existence of the CI tools chosen: Ant will be considered in use if there is one file called "build.xml", Buddy if there is a "buddy.yml" file, CircleCI if there is a ".circleci" file, CruiseControl if there is a "config.xml" file, Gradle if there is a "build.gradle" file, Hydra if there is a "hydra.conf" file, Jenkins if there is a "Jenkinsfile" file, Make if there is a "makefile" file, Maven if there is a "pom.xml" file, Sysphus if there is a "pkgconfig" file, TeamCity if there is a ".teamcity." file, TravisCI if there is a ".travis.yml" file, GitHub Workflows if there is a ".yml" file inside the project folder .github/workflows withing any job named "build".

- **CD tool**: CSE professionals frequently select tooling approaches to define and support CSE (J.Johanssen et al., 2018), so the CD tools mapping is relevant in this maturity stage. This work initially considered the CD tools based on a systematic review, where CD state of the art was reviewed classifying approaches and tools through stages *Configuration and Provisioning* and *CD Server* in a suggested deployment pipeline (M.Shahin; M.Babar; L.Zhu, 2017). However, as the systematic review considered

[4] <https://ant.apache.org/>
[5] <https://buddy.works/>
[6] <https://circleci.com/>
[7] <http://cruisecontrol.sourceforge.net/>
[8] <https://gradle.org/>
[9] <https://github.com/NixOS/hydra>
[10] <https://www.jenkins.io/>
[11] <https://www.gnu.org/software/make/>
[12] <https://maven.apache.org/>
[13] <https://doi.ieeecomputersociety.org/10.1109/CSMR.2007.49>
[14] <https://www.jetbrains.com/teamcity/>
[15] <https://travis-ci.org/>
[16] <https://docs.github.com/en/actions/using-workflows>

known best practices, it was necessary to adapt the output tools into the OSSP scenario. Thus, configuration, provisioning, and CD server tools that could not be fit in the OSSP context completely were eliminated from the systematic review study: Deployr, Hockey App, Jenkins (already mapped into CI), Microsoft Web Deploy. Likewise, to describe better OSSP context some CD tools were added: Docker (K.Eng; A.Hindle, 2021), Ansible and SaltStack (V.Sobeslav; A.Komarek, 2015). Hence, the final CI tools this work contemplates are: Ansible [17], Chef [18], Docker [19], Hiera [20], Puppet labs [21], SaltStack [22], Yum [23], GitHub Workflows [24].

From the process of collecting OSSP files represented by the entity *RepositoriesFiles* (described in the Subsection 5.3.1), for each project, there will be a search in each file path looking for patterns to prove the existence of the CD tools chosen: Ansible will be considered in use if there is one file called "ansible.cfg", Chef if there is a "config.rb" file, Docker if there is a "Dockerfile" file, Hiera if there is a "hiera.yaml" file, Puppet labs if there is a "puppet.conf" file, SaltStack if there is a "salt/%.conf" file, Yum if there is a "yum.conf" file, GitHub Workflows if there is a ".yml" file inside the project folder .github/workflows withing any job named "deploy" or "release".

4.Generate final results: From the previous milestone *3*, the metrics and classifications are stored in the MSc OSSP database (represented by the entity *ClassificationsMetricsReleases*, and *SDPClassificationsReleases* - described in the Subsection 5.3.1) for information and further analysis.

The final results related to the OSSP analysis are sent to a CSV file ("results.csv" in the folder outcome) writing the repository main data, metrics data, relevant dates, and classifications. The outcome file "results.csv" is the source of the results, discussions, and conclusions of this work. It contains the columns: *Id_Repo*, *Repo_Name*,

[17]<https://www.ansible.com/>
[18]<https://www.chef.io/>
[19]<https://www.docker.com/>
[20]<https://puppet.com/docs/puppet/7/hiera_intro.html>
[21]<https://puppet.com/>
[22]<https://docs.saltproject.io/>
[23]<http://yum.baseurl.org/>
[24]<https://docs.github.com/en/actions/using-workflows>

Repo_owner, *Main_Language*, *First_Date_CiCd*, *Min_Release_Date*, *Max_Release_Date*, *Delta_Days_First_Release_Until_CiCd*, *Delta_Days_Last_Release_After_CiCd*, *Count_Releases_Before_DateCiCd*, *Count_Releases_After_DateCiCd*, *Count_Releases_Before_CiCd_Low_Performers*, *Count_Releases_Before_CiCd_Medium_Performers*, *Count_Releases_Before_CiCd_High_Performers*, *Count_Releases_After_CiCd_Low_Performers*, *Count_Releases_After_CiCd_Medium_Performers*, *Count_Releases_After_CiCd_High_Performers*.

5.4 Final Considerations

This chapter presented the methodology for this project. The next chapter will show the results and discussions, given the outputs available described here as well.

CHAPTER 6

Results and discussions

6.1 Initial Considerations

This chapter first characterizes in Section 6.2 the results to answer the RQ using the methodology presented in the chapter before, and later in Section 6.3 provides discussions on how the findings of the result answered the RQ, as well as the main takeaways obtained.

6.2 Results

The first step to start the data extraction was defining the initial filtering to collect data from GitHub. In this particular case to get an updated dataset, this work adopts the popularity mean values Borges et al. work (H.Borges; A.Hora; M.Valente, 2016) proposes doing filters by stars and forks, so OSSP with more than 3440 stars and more than 532 forks were included.

As soon as the initial filtering was decided, the data extraction was started in July 2022 and, due to the high volume of data collected, this process had to be done progressively, as the main data source mined here GitHub has constraints set to protect its servers against excessive or abusive calls [1]. The Data Extraction and Analysis returned 1009 Repositories elective OSSP. Beyond the 1009 OSSP, this full run described here collected 725274 source-code files, 57338 Releases, 350103 Merged Pull requests, 37182 Branches, and 154 different types

[1] <https://docs.github.com/en/graphql/overview/resource-limitations>

of software development languages. The whole database is available with all inputs and outputs (D.Barros; F.Horita; I.Wiese, 2023b). Out of the 1009 OSSP collected 293 (29%) got everything that it takes (data and filtering) to be considered in the final results. The 716 (71%) OSSP who were not considered in the final results had a lack of data (e.g. Releases, Merged Pull Requests), or did not qualify in the filtering: more than fifty Releases, more than fifty Merged Pull Requests), and more than one year from the extraction date with a CI or CD tooling adoption.

Looking at the OSSP considered in the final results, after the filtering described in Section 5.3.2 step 3, in terms of software development languages the results got the majority of 24% of the projects in JavaScript as the main language, 23% TypeScript, 15% Go, 8% Python, 6% C++, and 5% Java. Table 5 shows the count of projects grouped by software development languages, and Figure 18 draws the same results to be visualized graphically.

Table 5: Software Development Languages used

Software Development Language	***Projects Count***
JavaScript	69
TypeScript	66
Go	43
Python	23
C++	17
Java	16
Rust	9
C, Ruby, Swift	6
C#, PHP	5
HTML, Kotlin, Shell, Vue	3
Haskell	2
Clojure, Elixir, Emacs Lisp, Julia, Lua, OCaml, Roff, V	1

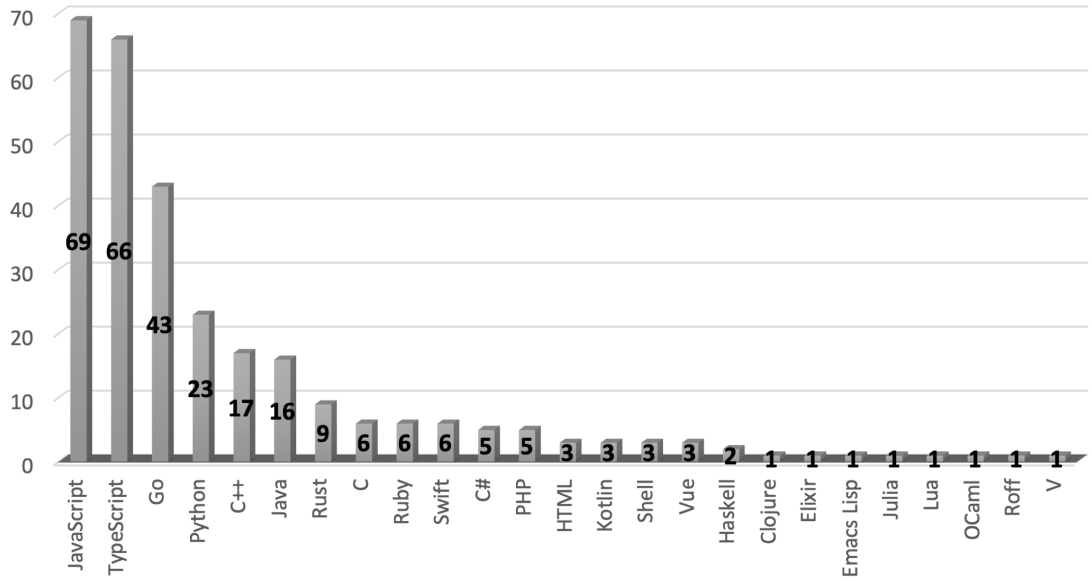

Figure 18: Software Development Languages used

Concerning the releases metrics results using the SDP Taxonomy, it is possible to see an evolution from Low Performers to Medium Performers when adopting CI/CD. In other words, adopting CI/CD allowed a process of decreasing the percentage count of Low Performers releases and increasing the percentage count of Medium Performers releases. Table 6 exhibits this situation where we had 78% of Low Performers releases before CI/CD against 69% after CI/CD. Likewise for Medium Performers where we had 22% of Medium Performers releases before CI/CD against 31% after CI/CD.

Table 6: Percentage Count of Releases considering CI/CD

SDP Taxonomy - Before CI/CD (N.Forsgren; J.Humble; G.Kim, 2018)		
Cluster	***Quantity***	***Percentage***
Low Performers	10757	78%
Medium Performers	2976	22%
High Performers	0	0%
SDP Taxonomy - With CI/CD (N.Forsgren; J.Humble; G.Kim, 2018)		
Cluster	***Quantity***	***Percentage***
Low Performers	27787	69%
Medium Performers	12319	31%
High Performers	0	0%

It is also possible to see graphically the improvements as well. Apart from some outliers in Low Performers with CI/CD, generally, the median release count is better in those that adopted CI/CD compared with those who did not adopt CI/CD. Figure 19 presents the Box Plots for all SDP Taxonomy clusters showing the values and median of the quartiles.

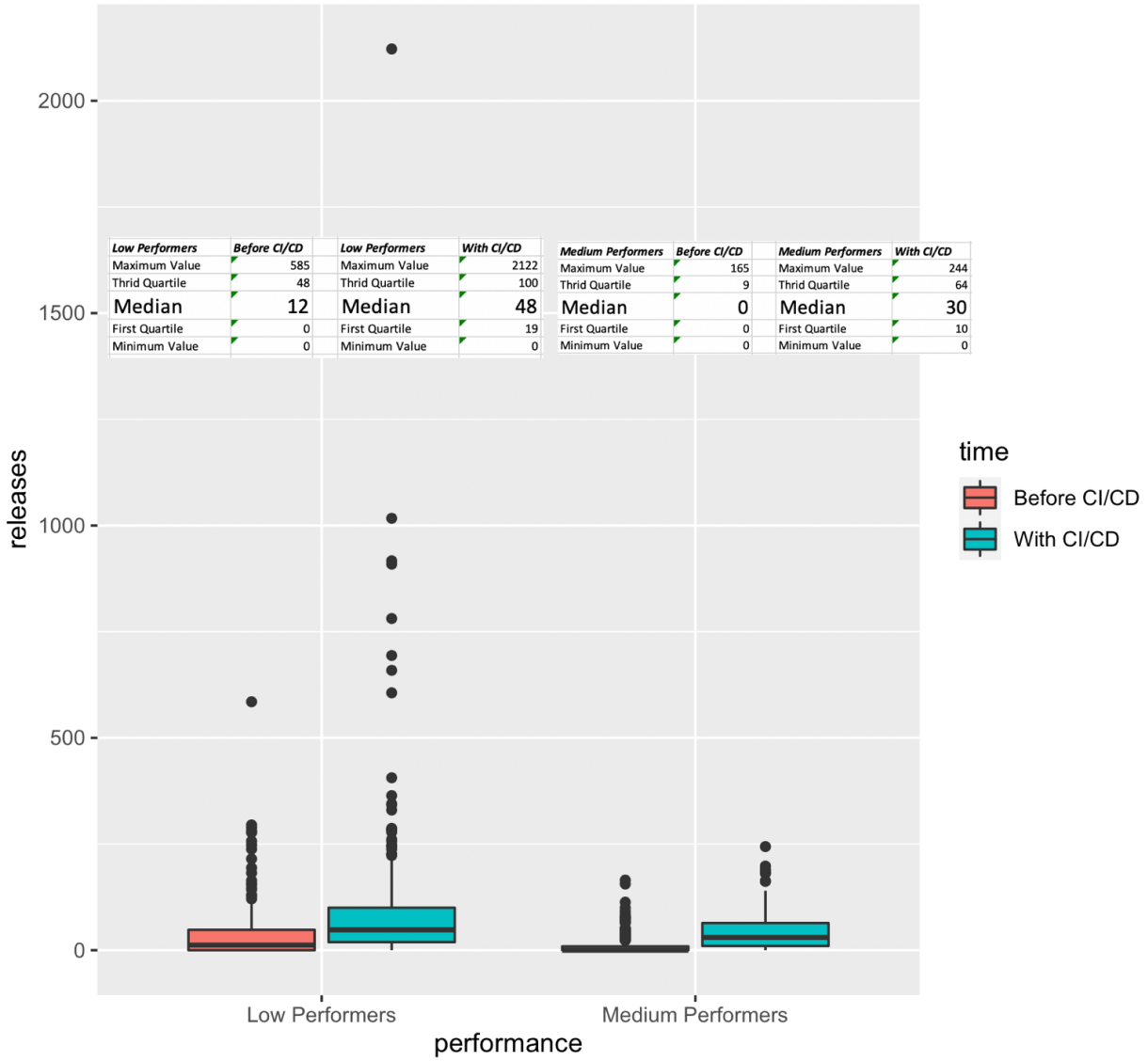

Figure 19: SDP Metrics Box Plot

Similarly, the same behavior happens when the analyzed results consider the top five languages covered by these results (JavaScript, TypeScript, Go, Python, C++). Apart from the same outliers Figure 19 showed, that it is possible to see improvements when adopting CI/CD. Figure 20 shows graphically the top five languages Box Plots for all SDP Taxonomy clusters showing the values and median of the quartiles.

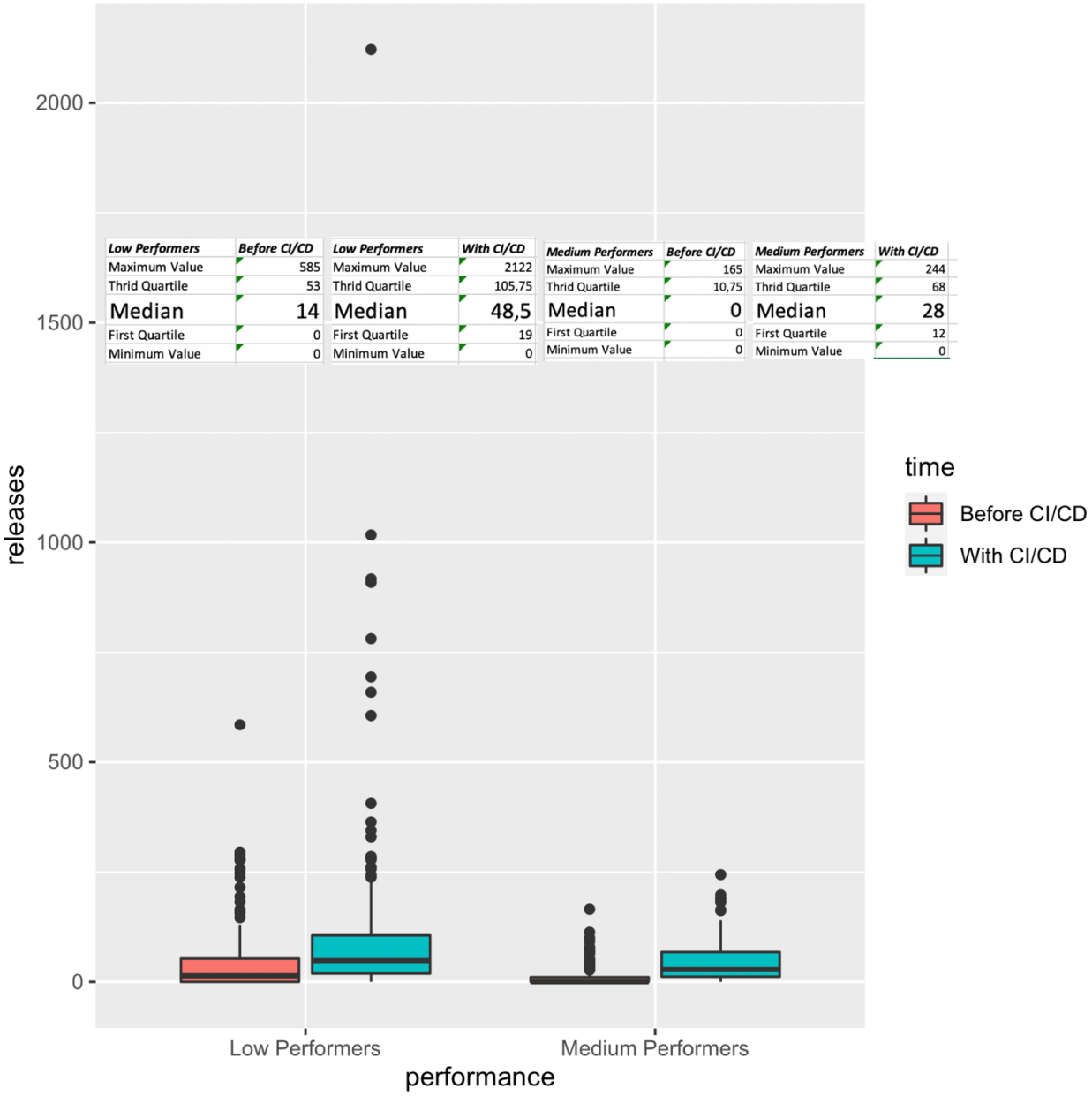

Figure 20: SDP Metrics Box Plot Top 5 Languages

The data results also allowed some tests (e.g. normality test, comparing two related groups, and computing the amount of difference between groups) to inspect individually the count of releases using the SDP Taxonomy (Low Performers and Medium Performers) before CI/CD and with CI/CD. Here, data results are analyzed through three statistical tests: Shapiro-Wilk normality test, Wilcoxon signed rank test with continuity correction, and Cliff's Delta.

The Shapiro-Wilk normality test (B.Yazici; S.Yolacan, 2007) verifies the normality of data. Testing each SDP Taxonomy considering before CI/CD and with CI/CD provided four groups to try: Low Performers before CI/CD, Low Performers with CI/CD, Medium Performers before CI/CD, and Medium Performers with CI/CD. High Performers were omitted as there were no OSSP classified in this cluster. Table 7 (considering all OSSP) and Table 8 (considering the top five languages) format the values according to the grouping described here, indicating that data results show a medium rate of ordered and standardized (W) with a small p-value (less than 0.05 and close to zero), therefore, data substantially deviate from a normal distribution.

Table 7: Shapiro-Wilk normality test considering CI/CD

SDP Taxonomy - Before CI/CD (N.Forsgren; J.Humble; G.Kim, 2018)		
Cluster	***W***	***p-value***
Low Performers	0.59758	< 2.2e-16
Medium Performers	0.51137	< 2.2e-16
SDP Taxonomy - With CI/CD (N.Forsgren; J.Humble; G.Kim, 2018)		
Cluster	***W***	***p-value***
Low Performers	0.43835	< 2.2e-16
Medium Performers	0.8557	7.14e-16

Table 8: Shapiro-Wilk normality test considering CI/CD Top 5 Languages

SDP Taxonomy - Before CI/CD (N.Forsgren; J.Humble; G.Kim, 2018)		
Cluster	***W***	***p-value***
Low Performers	0.60788	< 2.2e-16
Medium Performers	0.54861	< 2.2e-16
SDP Taxonomy - With CI/CD (N.Forsgren; J.Humble; G.Kim, 2018)		
Cluster	***W***	***p-value***
Low Performers	0.44205	< 2.2e-16
Medium Performers	0.85135	1.107e-13

The Wilcoxon signed rank test (D.Rey; M.Neuhäuser, 2011) with continuity correction was

used to compare related groups in the same SDP Taxonomy before CI/CD and after CI/CD (Low Performers before CI/CD, Low Performers with CI/CD, Medium Performers before CI/CD, and Medium Performers with CI/CD). Table 9 (considering all OSSP) and Table 10 (considering the top five languages) show the values according to the grouping described here, revealing that Low Performers groups have larger differences between the sampled groups compared with the Medium performers, given the V-statistic values. The p-values in this test were not considered as the previous test (Shapiro-Wilk normality test) already stated that data substantially deviate from a normal distribution.

Table 9: Wilcoxon signed rank test with continuity correction

SDP Taxonomy - Before CI/CD (N.Forsgren; J.Humble; G.Kim, 2018)	
Clusters	***V-statistic***
Low Performers - Before CI/CD X With CI/CD	11781
Medium Performers - Before CI/CD X With CI/CD	5636

Table 10: Wilcoxon signed rank test with continuity correction Top 5 Languages

SDP Taxonomy - Before CI/CD (N.Forsgren; J.Humble; G.Kim, 2018)	
Clusters	***V-statistic***
Low Performers - Before CI/CD X With CI/CD	6943
Medium Performers - Before CI/CD X With CI/CD	3356

The Cliff's Delta (G.Macbeth; E.Razumiejczyk; R.Ledesma, 2011) was used to compute the amount of difference between groups in the same SDP Taxonomy before CI/CD and after CI/CD (Low Performers before CI/CD, Low Performers with CI/CD, Medium Performers before CI/CD, and Medium Performers with CI/CD). Table 11 (considering all OSSP) and Table 12 (considering the top five languages) show the values according to the grouping described here, stating that the amount of difference between Low Performers groups is medium, against a large amount of difference between Low Performers where a change definitely happened in the transition between Before CI/CD to With CI/CD.

Table 11: Cliff's Delta

SDP Taxonomy - Before CI/CD (N.Forsgren; J.Humble; G.Kim, 2018)	
Clusters	***delta estimate***
Low Performers - Before CI/CD X With CI/CD	-0.4499179 (medium)
Medium Performers - Before CI/CD X With CI/CD	-0.6369789 (large)

Table 12: Cliff's Delta Top 5 Languages

SDP Taxonomy - Before CI/CD (N.Forsgren; J.Humble; G.Kim, 2018)	
Clusters	***delta estimate***
Low Performers - Before CI/CD X With CI/CD	-0.4245855 (medium)
Medium Performers - Before CI/CD X With CI/CD	-0.6298712 (large)

6.3 Discussions

The methodology and breadth of this work allowed, supported by the results collected, brought clarifications and alternatives on how popular OSSP behave when software delivery performance is the subject to be measured, as well as how to measure software delivery performance in these projects. This work findings can be used in the industry, where tools and private third-party companies can support SDP, together with in the open source world, where project data is also available to be collected and analyzed. Section 6.3.1 describes this work's implications for literature and practice, Section 6.3.2 examines the findings of the results and explains how they answered the RQ, and finally, Section 6.3.3 links how this work methodology can drive the SDP Improvements towards productivity.

6.3.1 Implications for literature and practice

The research contributions produced by this work include the following implications for literature and practice: A methodology to allow SDP Data Extraction and Analysis on most popular OSSP on a Release timeline basis; A public replication package where the methodology was applied in the popular OSSP context; A public dataset filled with both Data Extraction and Analysis from most popular OSSP.

A methodology to allow SDP Data Extraction and Analysis on most popular OSSP on a Release timeline basis started with a review process of SDP state-of-the-art, and the means to deal with data (extraction and analysis) resulting in several publications (D.Barros; F.Horita; D.Fantinato, 2020; D.Barros; F.Horita, 2020b; D.Barros; F.Horita, 2020a; D.Barros et al., 2021). In fact, collecting SDP data and providing analysis is not a novelty anymore. Chapter 3 shows related works where it is possible to see SDP Data Extraction and Analysis. However, this work goes further and offers a methodology for future research and practitioners, where SDP Data Extraction and Analysis happen on the most popular OSSP on a Release timeline basis. The relevance (in this case popularity) and the timeline basis (in this case Releases) were proved efficient, supported by the results collected on both POC and the final ones.

A public replication package where the methodology was applied in the popular OSSP context means an easy and tangible way to replicate this work and extend this research. Rather than restricting the scope to only a kind of project, or offering a commercial tool, this replication package [2] like its similar ones accepts community evolutions through Pull requests and bug reporting via Issues.

A public dataset filled with both Data Extraction and Analysis from the most popular OSSP is useful to understand these work findings as the structure and data are both available. Moreover, the public dataset (D.Barros; F.Horita; I.Wiese, 2023b) can be a

[2] <https://github.com/ddangelorb/gthbmining/tree/master/sdp>

baseline for future works as well, acting as a quick start for new researchers and related works.

6.3.2 RQ analysis: Considering Software Delivery Performance as a software delivery measurement model, how do popular Open Source Software Projects behave?

In view of the 29% of OSSP selected to be considered in the final results, the popularity filtering proposed allowed a proper analysis and confirmed initial assumptions. The best example is that the top languages in this work's findings met the open source software demand. For instance, it is possible to see JavaScript as the most popular programming language, supported by surveys too like the StackOverflow survey in May 2022 (StackOverflow, 2022). Besides, other popular programming languages are also matching the top ten final results with the StackOverflow survey such as TypeScript, Python, Java, C++, and C.

Looking at the release metrics results using the SDP Taxonomy it is possible to see numerically that the choice of the metrics was assertive, as well as the grouping by Releases. Where the metrics *Lead Time* and *Deployment frequency* are used to define properly the SDP Taxonomy, the CI/CD adoption established a clear division for improvement. The results confirmed the improvements stated here, decreasing the count of Low Performers releases and improving the count of Medium Performers releases. The findings outcomes showed that 78% of Low Performers releases before CI/CD dropped to 69% after CI/CD, along with 22% of Medium Performers releases before CI/CD increased to 31% after CI/CD.

Getting out the Releases outliers, it is also possible to see the trend stated in the previous paragraph where the CI/CD adoption decreased the count of Low Performers releases and improved the count of Medium Performers releases. For the Medium Performers, looking at an SDP Metrics Box Plot, the median releases count increased from zero (Before CI/CD) to thirty (After CI/CD), increasing all quartiles as well: first (minimum value), third, and fourth (maximum value). Similarly, looking at an SDP Metrics Box Plot, the top five languages confirmed this trend where the median releases count increased from zero (Before CI/CD) to twenty-eight (After CI/CD), increasing all quartiles as well: first (minimum value), third,

and fourth (maximum value).

For the purpose of understanding how data results values behave, in this case inspecting individually the count of releases using the SDP Taxonomy (Low Performers and Medium Performers) before CI/CD and with CI/CD, data results are analyzed through three statistical tests: **Shapiro-Wilk normality test**, **Wilcoxon signed rank test with continuity correction** and **Cliff's Delta**. **The Shapiro-Wilk normality test** stated that results data deviate from a normal distribution, and it is possible to see a high rate of ordered and standardized (W) data, specifically in Medium Performers with CI/CD, proving the improvements after CI/CD adoption; **The Wilcoxon signed rank test** showed more significant differences between the groups given the V-statistic values, however, it is possible to observe a significant drop in the differences in the values mainly in the Medium Performers. Thus, it is possible to intuitively state that the Medium Performers results are better than the Low Performers ones, as V-statistic values are lower. Therefore, the evolution from Low Performers provided a better numbers going to Medium performers; The **Cliff's Delta** test results also provided discussions beyond p-values and V-statistic interpretations shown previously. Here again, it is clear that the most relevant difference happened in the Medium Performers groups after CI/CD adoption. It means that an improvement happens when going from Low Performers to Medium Performer adoption CI/CD.

6.3.3 Developer Productivity improves SDP

Improvements in SDP cover several capabilities grouped by Architecture, Product and process, Lean management and monitoring, Cultural, and, of course, the group covered by this work CDE (N.Forsgren; J.Humble; G.Kim, 2018). Understanding that CDE capabilities use most of the time software developers as the main agents, it is possible to state that improving software developers' productivity should improve SDP, specifically the metrics chosen for this work.

Associating the findings and metrics of this work with the SPACE Framework (N.Forsgren et al., 2021), where important dimensions of developer productivity are taken into account, it is possible to confirm the the improvement in productivity when at least three SPACE Framework

dimensions increase altogether. Figure 21 shows three SPACE Framework dimensions, *Satisfaction with engineering system (e.g. CI/CD pipeline)*, *Story points shipped*, *Frequency of deployments*, mapped respectively to this work metrics as *CI/CD adoption*, *Lead Time*, and *Deployment frequency*. Therefore, here the SPACE Framework thesis that productivity cannot be reduced in a single dimension is ratified by this work's findings, where the SDP improvements come with all three metrics improvements together.

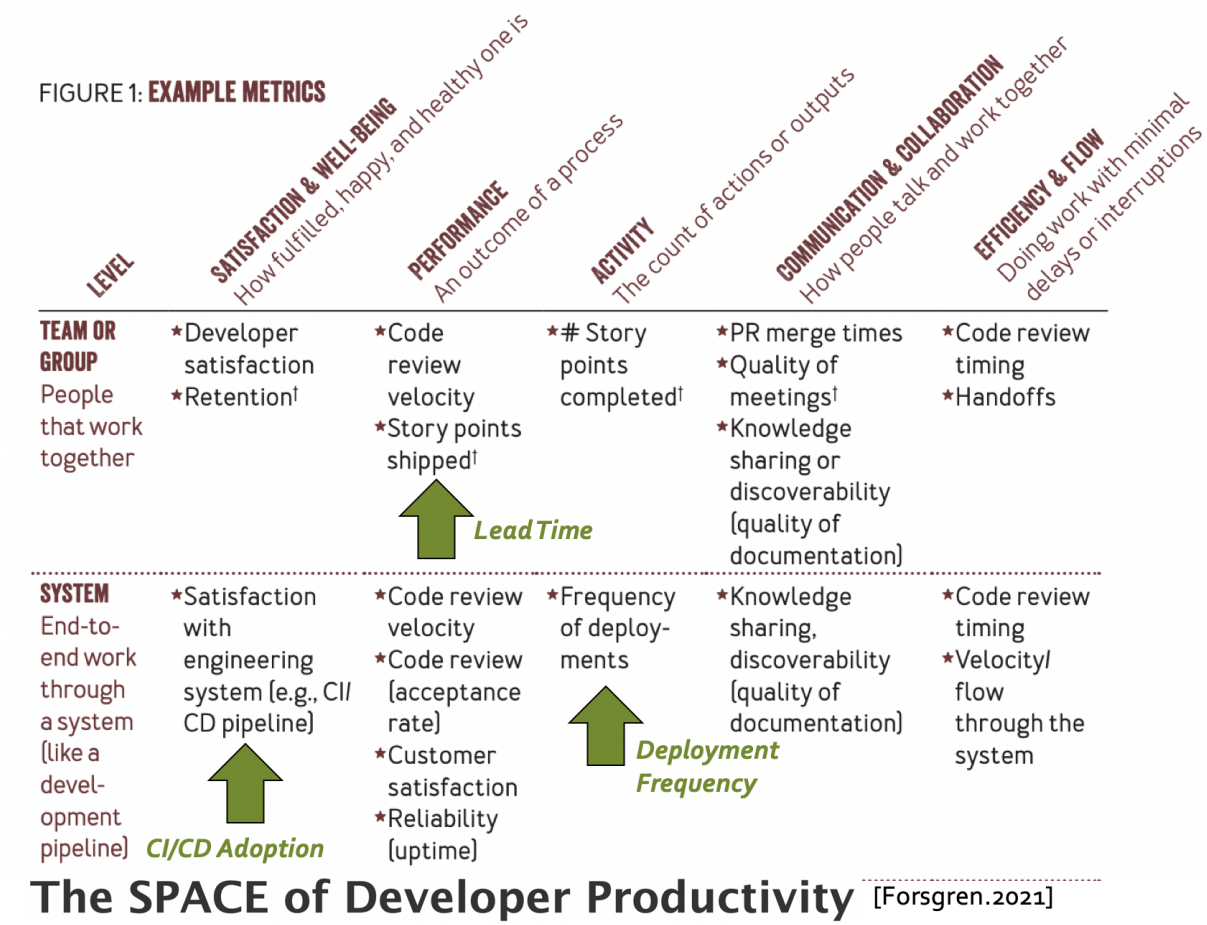

Figure 21: SPACE Framework mapped to this work metrics

6.4 Final Considerations

In this chapter the results of an entire run using the methodology presented were presented, as well as discussions on the analysis of the results, answers to the RQ, and findings analyses on how to improve SDP using productivity. The next chapter summarizes the conclusions

from this study, and opportunities for future works.

CHAPTER 7

Conclusions and future works

7.1 Initial Considerations

This chapter summarizes the conclusion of this work and refers to the research contribution made in Section 7.2, threats to validity in Section 7.3, and finally the recommendation of future work in Section 7.4.

7.2 Research contributions

Software delivery is an essential requirement today and the demand for it is constantly increasing. Rather than simply using an Agile development method to accelerate software delivery, with rapid releases, it is essential to measure the exact nature of the software delivery process and how it can be improved so that the current needs of customers can be met. However, measuring software delivery with the aid of maturity models, like CSE, might not be enough as these models set milestones to progress or incorporate generic formulas that may, or may not, suit all types of software projects. Capability models, like SDP, have thus emerged that are flexible enough to address the problem of software delivery measurement.

After carrying out an extensive literature review, publishing related articles, and creating a proof of concept to select the right metrics, this study employs a reliable and proven methodology to analyze SDP behavior in popular Open Source Software Projects on a Release

timeline basis through delivery metrics. A public replication package is available once the methodology has been fully implemented, to encourage reproducibility and future research.

The main relevance of this work is materializing, through the methodology and replication package, an automated manner to analyze SDP behavior in OSSP on a Release timeline basis through delivery metrics. The filtering, including popularity, is an input parameter, so this is the user's call to include or exclude more OSSP. Furthermore, this work collected automatically a relevant dataset, that relied on popularity, confirming previous statements only done by manual processes like surveys (N.Forsgren; J.Humble; G.Kim, 2018).

Additionally, with the aid of the dataset used in this study, it is also possible to link related works like the SPACE Framework (N.Forsgren et al., 2021) with our research findings, which provide evidence that improving SDP can be achieved by increasing at least 3 dimensions of the developers' productivity planned by the SPACE Framework.

7.3 Threats to validity

All the data needed to carry out this study was collected and referenced in previous sections. There have been no omissions or removal of data. However, during the course of this study, some points were raised which could cast doubts on the validity of the results. This is because predictive validity is essential to ensure the quality of the overall work. On the basis of the categories of threats set out by Zhou et al. (X.Zhou et al., 2016), the following potential risks can be outlined.

7.3.1 Internal Validity

Although this study used reliable methods to collect data, and the first version of the methodology was checked during the POC, the Data Extraction processing was automated and relied on the GitHub APIs available at the discretion of the author. This means that some OSSP data might have been missing or selected incorrectly. However, an attempt was made to fill this gap by double-checking 10% of the final dataset to avoid any data inconsistency.

Moreover, the methodology source code has resilient Data Extraction techniques, which means that partial load operations can be carried out after any stop processing. Since it was all covered by exception handling functions, any problem with bugs or data unavailability could be tracked easily.

7.3.2 External Validity

Some filtering was undertaken to ensure the data was obtained properly, which meant that the information retrieval module could provide accurate data for generating results and conducting further analyses. After the popularity, there was filtering by Releases, Merged Pull requests, and the First CI/CD Date. In this case, only the OSSP with more than fifty Releases, fifty Merged Pull requests, and more than one year with the first CI/CD data were taken into account. This implies that other important OSSP that did not comply with the filtering at the time of the Data Extraction process, may have been missed. Thus, it is not possible to generalize the conclusions that can be drawn for popular OSSP. Still, the outcomes of this study enable us to acquire knowledge about how the popular OSSP hosted on GitHub behaves.

7.4 Future works

Future research in this area should focus on extending the analysis of the OSSP behavior, by lowering the level of popularity so that it can cover more repositories. Given the fact that GitHub is likely to have more functionalities working together, such as DevOps pipelines and Issue tracking, new metrics can be added in future studies as well, starting with the SDP ones left over by this study: *MTTR* and *Change failure rate*.

SDP capability groups can be added to more extensive studies, either by means of automated Data Extraction, or manual data collection through surveys. These groups like Architecture, Product and process, Lean management and monitoring, or even Cultural capabilities, can be collected and would enhance the value of future conclusions. Moreover,

the collection of other SDP capabilities could also use new types of sources that this work did not explore, such as Docker Hub data through its API, or Travis CI and CircleCI APIs.

Last but not least, developer productivity, with the aid of the SPACE framework, could be broadly extended in future work as well. SPACE framework dimensions like *Communication and Collaboration* and *Efficiency and Flow* were not explored here at all, and could definitely make a significant contribution. It would also be worth investigating *Communication and Collaboration*, answers like Merged Pull requests extra information, how good the team meetings are, knowledge sharing, or even the quality of the documentation which can be extracted and used. With regard to *Efficiency and Flow*, outcomes such as the time for code reviews, hands-off, and velocity flow, can be extracted and used as well.

Bibliography

A.Sidky; J.Arthur; S.Bohner. A disciplined approach to adopting agile practices: the agile adoption framework. *Innovations in systems and software engineering*, Springer, v. 3, n. 3, p. 203–216, 2007.

A.Wiedemann et al. Research for practice: The devops phenomenon. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 62, n. 8, p. 44–49, jul. 2019. ISSN 0001-0782. Available on: <https://doi.org/10.1145/3331138>.

B.Adams; S.McIntosh. Modern release engineering in a nutshell – why researchers should care. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2016. v. 5, p. 78–90. Available on: <https://doi.org/10.1109/SANER.2016.108>.

Beck, K. *Extreme programming explained: embrace change.* : Addison-Wesley Professional, 2000.

B.Fitzgerald; K.Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, v. 123, p. 176–189, 2017. ISSN 0164-1212. Available on: <https://www.sciencedirect.com/science/article/pii/S0164121215001430>.

B.Vasilescu et al. Quality and productivity outcomes relating to continuous integration in github. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2015. (ESEC/FSE 2015), p. 805–816. ISBN 9781450336758. Available on: <https://doi.org/10.1145/2786805.2786850>.

B.Yazici; S.Yolacan. A comparison of various tests of normality. *Journal of Statistical Computation and Simulation*, Taylor Francis, v. 77, n. 2, p. 175–183, 2007. Available on: <https://doi.org/10.1080/10629360600678310>.

Claps, G. G.; Berntsson Svensson, R.; Aurum, A. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology*, v. 57, p. 21–31, 2015. ISSN 0950-5849. Available on: <https://www.sciencedirect.com/science/article/pii/S0950584914001694>.

C.Vassallo et al. Continuous delivery practices in a large financial organization. In: *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2016. p. 519–528. Available on: <https://doi.org/10.1109/ICSME.2016.72>.

C.Vassallo et al. A tale of ci build failures: An open source and a financial organization perspective. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2017. p. 183–193. Available on: <https://doi.org/10.1109/ICSME.2017.67>.

C.Vassallo et al. Context is king: The developer perspective on the usage of static analysis tools. In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2018. p. 38–49. Available on: <https://doi.org/10.1109/SANER.2018.8330195>.

D.Barros; F.Horita. Enhancing continuous delivery through release candidates. In: *IV Workshop NUVEM 2020 UFABC*. UFABC, 2020. (Nuvem '20 UFABC). Available on: <https://cutt.ly/CkoBbyP>.

D.Barros; F.Horita. Transformação digital através da mineração e aprendizagem devops: Um estudo no contexto de sistemas ciber-físicos. In: *Anais Estendidos do XVI Simpósio Brasileiro de Sistemas de Informação*. Porto Alegre, RS, Brasil: SBC, 2020. p. 30–34. ISSN 0000-0000. Available on: <https://sol.sbc.org.br/index.php/sbsi_estendido/article/view/13121>.

D.Barros; F.Horita; D.Fantinato. Data mining tool to discover devops trends from public repositories: Predicting release candidates with gthbmining.rc. In: *Proceedings of the 34th Brazilian Symposium on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2020. (SBES '20), p. 658–663. ISBN 9781450387538. Available on: <https://doi.org/10.1145/3422392.3422501>.

D.Barros; F.Horita; I.Wiese. *Proof of Concept database with inputs and outputs of the Master thesis: Analyzing Software Delivery Performance behavior in popular Open Source Software Projects on a Release timeline basis through delivery metrics*. Zenodo, 2023. Available on: <https://doi.org/10.5281/zenodo.7637693>.

D.Barros; F.Horita; I.Wiese. *Whole database with all inputs and outputs of the Master thesis: Analyzing Software Delivery Performance behavior in popular Open Source Software Projects on a Release timeline basis through delivery metrics*. Zenodo, 2023. Available on: <https://doi.org/10.5281/zenodo.7637650>.

D.Barros et al. A mining software repository extended cookbook: Lessons learned from a literature review. In: *Proceedings of the 35th Brazilian Symposium on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2021. (SBES '21). Available on: <https://doi.org/10.1145/3474624.3474627>.

D.Cohen; M.Lindval; P.Costa. An introduction to agile methods. In: . Elsevier, 2004, (Advances in Computers, v. 62). p. 1–66. Available on: <https://www.sciencedirect.com/science/article/pii/S0065245803620012>.

D.Costa et al. An empirical study of delays in the integration of addressed issues. In: *2014 IEEE International Conference on Software Maintenance and Evolution*. 2014. p. 281–290.

D.Costa et al. The impact of rapid release cycles on the integration delay of fixed issues. *Empirical Software Engineering*, Springer, v. 23, n. 2, p. 835–904, 2018. Available on: <https://doi.org/10.1007/s10664-017-9548-7>.

D.Costa et al. The impact of switching to a rapid release cycle on the integration delay of addressed issues: An empirical study of the mozilla firefox project. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2016. (MSR '16), p. 374–385. ISBN 9781450341868. Available on: <https://doi.org/10.1145/2901739.2901764>.

D.Grajales et al. Enabling participatory flood monitoring through cloud services. In: *ISCRAM 2022 Conference Proceedings - 19th International Conference on Information Systems for Crisis Response and Management*. Agder county, Norway: University of Agder, 2022. (Iscram 2022), p. 213–223. Available on: <http://idl.iscram.org/files/diegofabianpajaritograjales/2022/2411_DiegoFabianPajaritoGrajales_etal2022.pdf>.

D.Kini. Global project management - not business as usual. *Journal of Management in Engineering*, v. 16, n. 6, p. 29–33, 2000. Available on: <https://doi.org/10.1061/(ASCE)0742-597X(2000)16:6(29)>.

D.Rey; M.Neuhäuser. Wilcoxon-signed-rank test. In: _____. *International Encyclopedia of Statistical Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 1658–1659. ISBN 978-3-642-04898-2. Available on: <https://doi.org/10.1007/978-3-642-04898-2_616>.

E.Kula et al. Characterizing rapid releases in a large banking company: A case study. In: *BENEVOL*. 2018. p. 56–60. Available on: <http://ceur-ws.org/Vol-2361/short14.pdf>.

E.Kula et al. Releasing fast and slow: An exploratory case study at ing. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2019. (ESEC/FSE 2019), p. 785–795. ISBN 9781450355728. Available on: <https://doi.org/10.1145/3338906.3338978>.

E.Kula et al. Factors affecting on-time delivery in large-scale agile software development. *IEEE Transactions on Software Engineering*, p. 1–1, 2021. Available on: <https://doi.org/10.1109/TSE.2021.3101192>.

E.Laukkanen et al. Bottom-up adoption of continuous delivery in a stage-gate managed software organization. In: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: Association for Computing Machinery, 2016. (ESEM '16). ISBN 9781450344272. Available on: <https://doi.org/10.1145/2961111.2962608>.

F.Khomh et al. Do faster releases improve software quality? an empirical case study of mozilla firefox. In: *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. 2012. p. 179–188. Available on: <https://doi.org/10.1109/MSR.2012.6224279>.

F.Zampetti et al. How open source projects use static code analysis tools in continuous integration pipelines. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 2017. p. 334–344. Available on: <https://doi.org/10.1109/MSR.2017.2>.

G.Destro; B.França. Mining software repositories for the characterization of continuous integration and delivery. In: *Proceedings of the 34th Brazilian Symposium on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2020. (SBES '20), p. 664–669. ISBN 9781450387538. Available on: <https://doi.org/10.1145/3422392.3422503>.

GitHub. *GitHub Octoverse*. 2022. Available on: <https://octoverse.github.com/>.

G.Macbeth; E.Razumiejczyk; R.Ledesma. Cliff's delta calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica*, Pontificia Universidad Javeriana, v. 10, n. 2, p. 545–555, 2011. Available on: <http://www.scielo.org.co/pdf/rups/v10n2/v10n2a18.pdf>.

G.von Krogh; S.Spaeth. The open source software phenomenon: Characteristics that promote research. *The Journal of Strategic Information Systems*, v. 16, n. 3, p. 236–253, 2007. ISSN 0963-8687. Available on: <https://www.sciencedirect.com/science/article/pii/S096386870700025X>.

H.Borges; A.Hora; M.Valente. Understanding the factors that impact the popularity of github repositories. In: *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2016. p. 334–344. Available on: <https://doi.org/10.1109/ICSME.2016.31>.

Hemmati, H. et al. The msr cookbook: Mining a decade of research. In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. 2013. p. 343–352. ISSN 2160-1860. Available on: <https://doi.org/10.1109/MSR.2013.6624048>.

H.Kniberg. *Spotify engineering culture*. 2014. Available on: <https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1>.

H.Olsson; H.Alahyari; J.Bosch. Climbing the "stairway to heaven" – a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In: *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. 2012. p. 392–399. Available on: <https://doi.org/10.1109/SEAA.2012.54>.

Humble, J.; Molesky, J. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, v. 24, n. 8, p. 6, 2011. Available on: <https://www.uio.no/studier/emner/matnat/ifi/IN5430/v20/pensumliste/readings/humblemolesky_2011_devops.pdf>.

InovaUFABC. *III Congresso Prêmio UFABC de Inovação*. 2021. Available on: <https://www.youtube.com/watch?v=8zlrvsTWQO8>.

J.Bernardo; D.Costa; U.Kulesza. Studying the impact of adopting continuous integration on the delivery time of pull requests. In: *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. 2018. p. 131–141. Available on: <https://doi.org/10.1145/3196398.3196421>.

J.Humble; D.Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. : Pearson Education, 2010.

J.Johanssen et al. Practitioners' eye on continuous software engineering: An interview study. In: *Proceedings of the 2018 International Conference on Software and System Process*. New York, NY, USA: Association for Computing Machinery, 2018. (ICSSP '18), p. 41–50. ISBN 9781450364591. Available on: <https://doi.org/10.1145/3202710.3203150>.

J.Rubin; M.Rinard. The challenges of staying together while moving fast: An exploratory study. In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 2016. p. 982–993. Available on: <https://doi.org/10.1145/2884781.2884871>.

Júnior, P. S.; M.Barcellos; F.Ruy. Tell me: Am i going to heaven? a diagnosis instrument of continuous software engineering practices adoption. In: *Evaluation and Assessment in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2021. (EASE 2021), p. 30–39. ISBN 9781450390538. Available on: <https://doi.org/10.1145/3463274.3463324>.

Karvonen, T. et al. The crusoe framework: a holistic approach to analysing prerequisites for continuous software engineering. In: Springer. *International Conference on Product-Focused Software Process Improvement*. 2016. p. 643–661. Available on: <https://doi.org/10.1007/978-3-319-49094-6_52>.

K.Eng; A.Hindle. Revisiting dockerfiles in open source software over time. In: *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 2021. p. 449–459. Available on: <https://doi.org/10.1109/MSR52588.2021.00057>.

K.Silva; F.Horita. Compreendendo comportamentos emergentes em sistemas-de-sistemas por meio de simulação de software. In: *Anais Estendidos do XVII Simpósio Brasileiro de Sistemas de Informação (Anais Estendidos do SBSI 2021)*. Sociedade Brasileira de Computação (SBC), 2021. Available on: <https://doi.org/10.5753/sbsi.2021.15362>.

L.Lwakatare et al. Towards devops in the embedded systems domain: Why is it so hard? In: *2016 49th Hawaii International Conference on System Sciences (HICSS)*. 2016. p. 5437–5446. Available on: <https://doi.org/10.1109/HICSS.2016.671>.

L.Lwakatare et al. Devops in practice: A multiple case study of five companies. *Information and Software Technology*, v. 114, p. 217–230, 2019. ISSN 0950-5849. Available on: <https://www.sciencedirect.com/science/article/pii/S0950584917302793>.

M.Barcellos. Towards a framework for continuous software engineering. In: *Proceedings of the 34th Brazilian Symposium on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2020. (SBES '20), p. 626–631. ISBN 9781450387538. Available on: <https://doi.org/10.1145/3422392.3422469>.

M.Beedle et al. *Manifesto for Agile Software Development*. 2001. Available on: <https://agilemanifesto.org/>.

M.Chrissis; M.Konrad; S.Shrum. *CMMI for development: guidelines for process integration and product improvement*. : Pearson Education, 2011.

M.Hilton et al. Usage, costs, and benefits of continuous integration in open-source projects. In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2016. (ASE 2016), p. 426–437. ISBN 9781450338455. Available on: <https://doi.org/10.1145/2970276.2970358>.

M.Mäntylä et al. On rapid releases and software testing: a case study and a semi-systematic literature review. *Empirical Software Engineering*, Springer, v. 20, n. 5, p. 1384–1425, 2015. Available on: <https://doi.org/10.1007/s10664-014-9338-4>.

M.Michlmayr; B.Fitzgerald; K.Stol. Why and how should open source projects adopt time-based releases? *IEEE Software*, v. 32, n. 2, p. 55–63, 2015. Available on: <https://doi.org/10.1109/MS.2015.55>.

M.Sallin et al. Measuring software delivery performance using the four key metrics of devops. In: Springer, Cham. *International Conference on Agile Software Development*. 2021. p. 103–119. Available on: <https://doi.org/10.1007/978-3-030-78098-2_7>.

M.Shahin; M.Babar; L.Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, v. 5, p. 3909–3943, 2017. Available on: <https://doi.org/10.1109/ACCESS.2017.2685629>.

M.Shahin et al. Beyond continuous delivery: An empirical investigation of continuous deployment challenges. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2017. p. 111–120. Available on: <https://doi.org/10.1109/ESEM.2017.18>.

M.Tufano et al. When and why your code starts to smell bad. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. 2015. v. 1, p. 403–414. Available on: <https://doi.org/10.1109/ICSE.2015.59>.

N.Fogelström et al. The impact of agile principles on market-driven software product development. *Journal of Software Maintenance and Evolution: Research and Practice*, v. 22, n. 1, p. 53–80, 2010. Available on: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spip.420>.

N.Forsgren; J.Humble; G.Kim. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. : IT Revolution, 2018.

N.Forsgren et al. A taxonomy of software delivery performance profiles: Investigating the effects of devops practices. p. 1–6, 2020. Available on: <https://digitalscholarship.unlv.edu/met_fac_articles/118/>.

N.Forsgren et al. The space of developer productivity: There's more to it than you think. *Queue*, Association for Computing Machinery, New York, NY, USA, v. 19, n. 1, p. 20–48, feb 2021. ISSN 1542-7730. Available on: <https://doi.org/10.1145/3454122.3454124>.

N.Forsgren et al. Dora platform: Devops assessment and benchmarking. In: A.Maedche; Brocke, J.; A.Hevner (Ed.). *Designing the Digital Transformation*. Cham: Springer

International Publishing, 2017. p. 436–440. ISBN 978-3-319-59144-5. Available on: <https://doi.org/10.1007/978-3-319-59144-5_27>.

O.Ozcan-Top; O.Demirörs. Assessment of agile maturity models: A multiple case study. In: T.Woronowicz et al. (Ed.). *Software Process Improvement and Capability Determination*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 130–141. ISBN 978-3-642-38833-0. Available on: <https://doi.org/10.1007/978-3-642-38833-0_12>.

P.Abrahamsson et al. New directions on agile methods: a comparative analysis. In: *25th International Conference on Software Engineering, 2003. Proceedings.* 2003. p. 244–254. Available on: <https://doi.org/10.1109/ICSE.2003.1201204>.

P.Leon; F.Horita. On the modernization of systems for supporting digital transformation: A research agenda. In: *XVII Brazilian Symposium on Information Systems*. New York, NY, USA: Association for Computing Machinery, 2021. (SBSI 2021). ISBN 9781450384919. Available on: <https://doi.org/10.1145/3466933.3466976>.

P.Rodríguez et al. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, v. 123, p. 263–291, 2017. ISSN 0164-1212. Available on: <https://www.sciencedirect.com/science/article/pii/S0164121215002812>.

R.Fontana et al. Processes versus people: How should agile software development maturity be defined? *Journal of Systems and Software*, v. 97, p. 140–155, 2014. ISSN 0164-1212. Available on: <https://www.sciencedirect.com/science/article/pii/S0164121214001587>.

R.Júnior et al. Flying over brazilian organizations with zeppelin: A preliminary panoramic picture of continuous software engineering. In: *Proceedings of the XXXVI Brazilian Symposium on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2022. (SBES '22), p. 279–288. ISBN 9781450397353. Available on: <https://doi.org/10.1145/3555228.3555234>.

R.Kallis et al. Predicting issue types on github. *Science of Computer Programming*, v. 205, p. 102598, 2021. ISSN 0167-6423. Available on: <https://www.sciencedirect.com/science/article/pii/S0167642320302069>.

S.Joshi; S.Chimalakonda. Rapidrelease - a dataset of projects and issues on github with rapid releases. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 2019. p. 587–591. Available on: <https://doi.org/10.1109/MSR.2019.00088>.

S.Mohammad. Continuous integration and automation. *International Journal of Creative Research Thoughts (IJCRT), ISSN*, p. 938–945, 2016. Available on: <https://ssrn.com/abstract=3655567>.

S.Neely; S.Stolt. Continuous delivery? easy! just change everything (well, maybe it is not that easy). In: *2013 Agile Conference*. 2013. p. 121–128. Available on: <https://doi.org/10.1109/AGILE.2013.17>.

S.Stavru. A critical examination of recent industrial surveys on agile method usage. *Journal of Systems and Software*, v. 94, p. 87–97, 2014. ISSN 0164-1212. Available on: <https://doi.org/10.1016/j.jss.2014.03.041>.

StackOverflow. *StackOverflow Developer Survey*. 2022. Available on: <https://survey.stackoverflow.co/2022/>.

Store, J. *Qualities and Issues of Branching: A Method Proposal for Formulating a Branching Strategy*. University of Helsinki, 2020. Available on: <http://urn.fi/URN:NBN:fi:hulib-202012084714>.

T.Bissyandé et al. Got issues? who cares about it? a large scale investigation of issue trackers from github. In: *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 2013. p. 188–197. Available on: <https://doi.org/10.1109/ISSRE.2013.6698918>.

T.Rausch et al. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 2017. p. 345–355. Available on: <https://doi.org/10.1109/MSR.2017.54>.

Veldhoven, Z.; J.Vanthienen. Designing a comprehensive understanding of digital transformation and its impact. In: *Bled eConference*. 2019. p. 22. Available on: <https://doi.org/10.18690/978-961-286-280-0.39>.

V.Sobeslav; A.Komarek. Opensource automation in cloud computing. In: W.Wong (Ed.). *Proceedings of the 4th International Conference on Computer Engineering and Networks*. Cham: Springer International Publishing, 2015. p. 805–812. ISBN 978-3-319-11104-9. Available on: <https://doi.org/10.1007/978-3-319-11104-9_93>.

X.Zhou et al. A map of threats to validity of systematic literature reviews in software engineering. In: *Proceedings of the 23rd Asia-Pacific Software Engineering Conference (APSEC)*. 2016. p. 153–160. Available on: <https://doi.org/10.1109/APSEC.2016.031>.

Y.Jiang. *Mining Software Repositories for Release Engineers - Empirical Studies on Integration and Infrastructures-as-Code*. Tese (Doutorado) — École Polytechnique de Montréal, 2016. Available on: <https://publications.polymtl.ca/2255/>.

Y.Yu et al. Initial and eventual software quality relating to continuous integration in github. *arXiv preprint arXiv:1606.00521*, 2016. Available on: <https://doi.org/10.48550/arXiv.1606.00521>.

Y.Zhang et al. One size does not fit all: An empirical study of containerized continuous deployment workflows. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2018. (ESEC/FSE 2018), p. 295–306. ISBN 9781450355735. Available on: <https://doi.org/10.1145/3236024.3236033>.

Z.Khalil et al. On the impact of release policies on bug handling activity: A case study of eclipse. *Journal of Systems and Software*, v. 173, p. 110882, 2021. ISSN 0164-1212. Available on: <https://www.sciencedirect.com/science/article/pii/S0164121220302727>.

APPENDIX A

Publishing, Participation and Academic Services

Following in this appendix, a list of relevant published works, events the author was engaged in, and academic services.

Published works

- As the first author presented the paper **"Data Mining Tool to Discover DevOps Trends from Public Repositories: Predicting Release Candidates with Gthb-mining.Rc"** (D.Barros; F.Horita; D.Fantinato, 2020) at *XI Congresso Brasileiro de Software: Teoria e Prática (CBSoft SBES 2020)* in Tools Track. *October 2020.*

- As the first author presented the paper **"Transformação Digital através da mineração e aprendizagem DevOps: Um estudo no contexto de sistemas ciberfísicos"** (D.Barros; F.Horita, 2020b) at *XIV Simpósio Brasileiro de Sistemas de Informação (SBSI 2020 - WTDSI)* in track: Thesis and Dissertation Workshop on Information Systems. *November 2020.*

- As the first author presented the short paper **"Enhancing Continuous Delivery through Release Candidates"** (D.Barros; F.Horita, 2020a) at *IV Workshop NUVEM 2020 UFABC. November 2020.*

- As the first author presented the paper **"A Mining Software Repository Extended Cookbook: Lessons learned from a literature review"** (D.Barros et al., 2021)

at *XII Congresso Brasileiro de Software: Teoria e Prática (CBSoft SBES 2021)* in Research Track. *September 2021.*

- Co-authoring the paper **"Enabling Participatory Flood Monitoring Through Cloud Services'** (D.Grajales et al., 2022) at *19th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2022)* in Applications, Tools and Components for Crisis Management Track. *May 2022.*

Participation on Events

- Attended the **XI Congresso Brasileiro de Software: Teoria e Prática (CBSoft SBES 2020)**. *October 2020.*
- Attended the **XIV Simpósio Brasileiro de Sistemas de Informação (SBSI 2020 - WTDSI)**. *November 2020.*
- Attended the **IV Workshop NUVEM 2020 UFABC**. *November 2020.*
- Attended the **III Congresso Prêmio UFABC de Inovação** and was honored as the inventor of the gthbmining.rc technology (D.Barros; F.Horita; D.Fantinato, 2020; InovaUFABC, 2021). *September 2021.*
- Attended the **XII Congresso Brasileiro de Software: Teoria e Prática (CBSoft SBES 2021)**, where his work "A Mining Software Repository Extended Cookbook: Lessons learned from a literature review" (D.Barros et al., 2021) won an award as a distinguished paper in Research Track. *September 2021.*

Academic Services

- Volunteer reviewer in **VII Escola Regional de Computação Ceará, Maranhão, Piauí (ERCEMAPI 2019)**. *August 2019.*
- Volunteer reviewer in **IX Brazilian Workshop on Social Network Analysis and Mining (CSBC 2020 - BraSNAM)**. *June 2020.*

- Volunteer reviewer in **XI Congresso Brasileiro de Software: Teoria e Prática (CBSoft 2020 - MSSiS)**. *September 2020.*
- Volunteer reviewer in **XIX Simpósio Brasileiro de Qualidade de Software (SBQS 2020 - Research Track)**. *October 2020.*
- Volunteer reviewer in **III Workshop de Modelagem e Simulação de Sistemas Intensivos em Software (CBSoft 2021 - III MSSiS)**. *July 2021.*
- Volunteer reviewer in **XVIII Simpósio Brasileiro de Sistemas de Informação (SBSI 2022)**. *March 2022.*

APPENDIX B

Extra tooling mapped

In addition to the related works described in Chapter 3, there are several tools (commercial and open source) this work mapped that can analyze software delivery performance as well. Although these tools do not include behavior analysis in OSSP, as this work does, and were found mostly in the grey literature, they were used to enrich this work methodology and comparison. Here is the extra tooling mapped: *Oobeya* [1], *Swarmia* [2], *Linearb* [3], *Faros.ia* [4], *Plandek* [5], *Sleuth* [6], *Insightly* [7], *Haystack* [8], *Pluralsight* [9], *Pulse* [10], *Propelo* [11], *CodeFresh* [12], *LeanIX* [13], *Jellyfish* [14], *Keypup* [15], *Waydev* [16], *Hatica* [17], *Klera* [18], *Harness* [19],

[1] <https://oobeya.io/>
[2] <https://www.swarmia.com/>
[3] <https://linearb.io/>
[4] <https://www.faros.ai/>
[5] <https://plandek.com/>
[6] <https://www.sleuth.io/>
[7] <https://www.insightly.ai/>
[8] <https://www.usehaystack.io/>
[9] <https://www.pluralsight.com/>
[10] <https://www.pulse.codacy.com/>
[11] <https://www.propelo.ai/>
[12] <https://codefresh.io/>
[13] <https://www.leanix.net/>
[14] <https://jellyfish.co/platform/devops-metrics/>
[15] <https://www.keypup.io/>
[16] <https://waydev.co/>
[17] <https://www.hatica.io/>
[18] <https://www.klera.io/use-case/dora-metrics-monitoring/>
[19] <https://harness.io/blog/dora-metrics>

CTO.AI [20], *Opsera* [21], *Echoes HQ* [22], *samsmithnz/DevOpsMetrics* [23], *OkayHQ/ee-handbook* [24], *mikaelvesavuori/dorametrix* [25], *mikaelvesavuori/dorametrix-action* [26], *mikaelvesavuori/demo-dorametrix-action* [27], *mbagsik00/ bitbucket-dora-metrics* [28], mikaelvesavuori/dorametrix-pipe [29], *mikaelvesavuori/demo-dorametrix-pipe* [30].

[20]<https://cto.ai/insights>
[21]<https://www.opsera.io/>
[22]<https://www.echoeshq.com/for/engineers-and-builders>
[23]<https://github.com/samsmithnz/DevOpsMetrics>
[24]<https://github.com/OkayHQ/ee-handbook>
[25]<https://github.com/mikaelvesavuori/dorametrix>
[26]<https://github.com/mikaelvesavuori/dorametrix-action>
[27]<https://github.com/mikaelvesavuori/demo-dorametrix-action>
[28]<https://github.com/mbagsik00/bitbucket-dora-metrics>
[29]<https://github.com/mikaelvesavuori/dorametrix-pipe>
[30]<https://github.com/mikaelvesavuori/demo-dorametrix-pipe>