Simulation of collaboration networks in software development

José Miguel Pereira Gomes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Francisco João Duarte Cordeiro Correia dos Santos
Prof. Antonio Manuel Ferreira Rito da Silva

Examination Committee

Chairperson: Prof. Nuno Miguel Carvalho dos Santos
Supervisor: Prof. Francisco João Duarte Cordeiro Correia dos Santos
Member of the Committee: Prof. Miguel Carlos Pacheco Afonso Goulão

November 2022
Acknowledgments

To my parents, who supported me in every possible manner, you are the key to this achievement. To my grandparents, for their continuous encouragement and care.

To my supervisors, Prof. Antônio Rito Silva and Prof. Francisco Santos, for their advice, ideas and continuous assistance, which lead this Thesis into existence.

To David and Rodrigo, for their review of this Thesis.
To my colleagues, for their support and splendid discussions throughout the course, specially Telmo, for his comradeship during all these years.

To all my friends, for their continuous motivation, companionship, and endless adventures.

Thank you. It could not be possible without you all. Hope it will make you all proud.
Abstract

Software development creates a social system made of complex networks of exchange and cooperation. Little is known, however, about their structural patterns and dynamics. Here, we characterize the emergent collaboration networks among developers through their interactions with files. Resorting to several open-source codebases, we show that these networks exhibit a power-law dependence on their connectivities. We show that this property emerges from multiple sources: Developers choosing files to work on according to a preferential attachment rule, a power law distribution on the number of commits by a developer, and specific rules to select which developer should commit next. The impact of these principles is shown to be qualitatively similar across codebases. Our results suggest the existence of scaling laws in software development, and that these patterns are explainable through simple principles, even without other key concepts in software development, such as teams or modules.

Keywords

Software Development; Collaboration; Complex Networks; Emergent Patterns; Simulation; Preferential Attachment
Resumo

O desenvolvimento de software cria um sistema social feito de redes complexas de comunicação e cooperação. Pouco se sabe sobre os padrões estruturais e dinâmicas destas redes. Neste trabalho, nós caracterizamos as redes de colaboração entre developers, emergentes das suas interações com ficheiros. Através de várias codebases open-source, nós mostramos que estas redes exibem uma dependência de lei de potência nas suas conectividades. Nós mostramos que esta propriedade surge de várias fontes: Os developers escolhem ficheiros para trabalhar de acordo com uma regra de ligação preferencial, uma distribuição de lei de potência no número de commits e regras específicas para escolher que developer deve fazer o próximo commit. É observado que o impacto destes princípios é qualitativamente semelhante em diversas codebases. Os nossos resultados sugerem a existência de regras de escala em desenvolvimento de software, e que estes padrões são explicados através de princípios simples, sem recorrer a outros conceitos chave em desenvolvimento de software, tal como equipas e módulos.

Palavras Chave

Desenvolvimento de Software; Colaboração; Redes Complexas; Padrões Emergentes; Simulação; Ligação preferencial
Contents

1 Introduction 2
  1.1 Introduction ................................................. 3
  1.2 Organization of the Document ................................. 4

2 Background 6
  2.1 Software Development in Version Control Systems .......... 7
  2.2 Network Science .............................................. 7
    2.2.1 Networks Characterization ............................... 7
    2.2.2 Real-world Networks Analysis ........................... 8
    2.2.3 Principles behind Real-world Networks Patterns .......... 10
    2.2.4 Impact of Networks Topologies ........................... 11

3 Related Work 12
  3.1 Relations between Developers ................................ 13
  3.2 Interactions between Developers and Files .................... 14
  3.3 Models that simulate the Development of Software ............ 15
  3.4 Validity of Research ........................................ 15
  3.5 Summary and Discussion ..................................... 16

4 Modelling Approach 18
  4.1 Approach .................................................. 19
  4.2 Networks ................................................... 19
    4.2.1 Network Construction ................................... 21
  4.3 Analysis Method ............................................. 25

5 Empirical Emergent Patterns 26
  5.1 Data Gathering ............................................. 27
  5.2 Collaboration Actors Patterns ............................... 27
  5.3 Collaboration Patterns ..................................... 30
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Simulation of the Emergent Patterns</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>6.1 Approach</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>6.2 Algorithm</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>6.3 Results</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>6.3.1 First Iteration</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>6.3.2 Second Iteration</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>6.3.3 Third Iteration</td>
<td>42</td>
</tr>
<tr>
<td>7</td>
<td>Discussion</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>7.1 Software Engineering Considerations</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>7.2 Threats to Validity</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>7.2.1 System Modelling</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>7.2.2 Data Gathering</td>
<td>57</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>8.1 Conclusions</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>8.2 Future Work</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>8.3 Thesis Artifacts</td>
<td>61</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>61</td>
</tr>
<tr>
<td>A</td>
<td>Simulations</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>A.1 First Iteration</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>A.2 Second Iteration</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>A.3 Complex Evaluation</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>A.4 Third Iteration</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>A.5 Collaboration per File Distributions</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>A.6 Codebase Aspects Evaluation</td>
<td>87</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Comparison between cumulative distributions of a power law distribution and an exponential distribution. ................................................................. 9

4.1 Networks resulting from commit history described in Table 4.1. ......................... 21

4.2 Example of a non-deterministic method rename. ............................................. 22

5.1 Representation and Distribution Statistical Analysis (DSA) of Developer To File Network’s Developers Weighted Degree Distribution (D2F/D/dist) for the distinct codebases regarding All Time (AT) Developer To File Network (D2F) networks. .................. 28

5.2 Representation and DSA of Developer To File Network’s Files Weighted Degree Distribution (D2F/F/dist) for the distinct codebases regarding AT D2F networks. .................. 29

5.3 Representation and DSA of File To File Network’s Weighted Degree Distribution (F2F/dist) for the distinct codebases regarding AT D2F networks. .................. 30

5.4 Representation and DSA of File Developer To Developer Network’s Weighted Degree Distribution (FD2D/dist) for the distinct codebases regarding AT File Developer To Developer Network (FD2D) networks. .................. 31

6.1 Representation and evaluation of the resulting Distribution To Analyze (Dist) from simulations $s^1_k$, with $k \in \{-, 0, 10, 15\}$, considering the Bitcoin codebase. ........................................ 40

6.2 Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 10, 15\}$, considering the Bitcoin codebase. ........................................ 41

6.3 newDevelopersUntilTimeT aspect comparison between timeline generated for $s^2_k$ and empirical timeline, concerning Bitcoin codebase. ................................. 42

6.4 Representation and evaluation of the resulting Dist from simulations $s^8_k$, with $k \in \{-, 0, 10, 15\}$, considering the Bitcoin codebase. ........................................ 46

6.5 Representation of Collaboration Done Per File Distribution (CollabPerFile/dist) resulting from simulations $s^8_k$, with $k \in \{-, 0, 10, 15\}$ in comparison with empirical CollabPerFile/dist, for Bitcoin codebase. ........................................ 47
6.6 Comparison between codebase aspects related with commits and developers of Codebase Aspects Evaluation (CBAspEval) between the empirical timeline of the Bitcoin codebase and the theoretical timeline generated with support from $s\text{Dists}_8$ for Bitcoin codebase. 48

6.7 Comparison between codebase aspects related with files and dates of CBAspEval between the empirical timeline of the Bitcoin codebase and the theoretical timeline generated with support from $s\text{Dists}_8$ for Bitcoin codebase. 49

7.1 Comparison, per developer, of its Weighted Degree (WD) in FD2D and Method Developer To Developer (MD2D) across the different analysed codebases. 56

A.1 Representation and evaluation of the resulting Dist from simulations $s^1_k$, with $k \in \{-, 0, 2.5, 5\}$, considering the Ansible codebase. 68

A.2 Representation and evaluation of the resulting Dist from simulations $s^1_k$, with $k \in \{-, 0, 5, 10\}$, considering the Kubernetes codebase. 69

A.3 Representation and evaluation of the resulting Dist from simulations $s^1_k$, with $k \in \{-, 0, 10, 15\}$, considering the Tensorflow codebase. 70

A.4 Representation and evaluation of the resulting Dist from simulations $s^1_k$, with $k \in \{-, 0, 2.5, 5\}$, considering the Terraform codebase. 71

A.5 Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 2.5, 5\}$, considering the Ansible codebase. 72

A.6 Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 5, 10\}$, considering the Kubernetes codebase. 73

A.7 Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 10, 15\}$, considering the Tensorflow codebase. 74

A.8 Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 2.5, 5\}$, considering the Terraform codebase. 75

A.9 Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 2.5, 5\}$, considering the Ansible codebase. 82

A.10 Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 5, 10\}$, considering the Kubernetes codebase. 83

A.11 Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 10, 15\}$, considering the Tensorflow codebase. 84

A.12 Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 2.5, 5\}$, considering the Terraform codebase. 85

A.13 Representation of CollabPerFile/dist resulting from simulations $s^8_k$, with $j = j_{\text{rcb}}$, and $k \in h_{\text{rcb}}$ in comparison with empirical CollabPerFile/dist, across different analysed codebases. 86
A.14 CBAspEval, per Real Codebase (RCB), between its empirical timeline and the theoretical
timeline generated in Listing 6.1 with support from $s_{Dist_{j,rcb}}$. . . . . . . . . . . . . . . . . 87
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Hypothetic commit history.</td>
<td>20</td>
</tr>
<tr>
<td>5.1</td>
<td>Characteristics of investigated codebases.</td>
<td>27</td>
</tr>
<tr>
<td>5.2</td>
<td>Percentage of D2F/D/dist DSA results for the distinct codebases regarding TL D2Fs.</td>
<td>28</td>
</tr>
<tr>
<td>5.3</td>
<td>Percentage of D2F/D/dist DSA results for the distinct codebases regarding TL D2Fs.</td>
<td>29</td>
</tr>
<tr>
<td>5.4</td>
<td>Percentage of F2F/dist DSA results for the distinct codebases regarding TL File to File Network (F2F)s.</td>
<td>30</td>
</tr>
<tr>
<td>5.5</td>
<td>Percentage of FD2D/dist DSA results for the distinct codebases regarding TL FD2Ds.</td>
<td>31</td>
</tr>
<tr>
<td>6.1</td>
<td>Complex Evaluation (CEval) of $s^j_k$, with $j \in [3-8]$ and $k \in {-0,0,10,15}$, for Bitcoin codebase.</td>
<td>45</td>
</tr>
<tr>
<td>6.2</td>
<td>CBAspEval between the empirical timeline of the Bitcoin codebase and the theoretical timeline generated in Listing 6.1 with support from $s_{Dists8}$ for Bitcoin codebase.</td>
<td>49</td>
</tr>
<tr>
<td>6.3</td>
<td>Comparison between the best inputs for each codebase and the empirical average changes per file, for different codebases.</td>
<td>50</td>
</tr>
<tr>
<td>7.1</td>
<td>Top 4 most changed files for the various analysed codebases and their functionalities.</td>
<td>53</td>
</tr>
<tr>
<td>7.2</td>
<td>Top 4 most coupled files for the various analysed codebases and their functionalities.</td>
<td>54</td>
</tr>
<tr>
<td>7.3</td>
<td>Power law fitting to the relation between the WDs per developer in FD2D and MD2D across the different analysed codebases.</td>
<td>57</td>
</tr>
<tr>
<td>A.1</td>
<td>CEval of $s^j_k$, with $j \in [3-8]$ and $k \in {-0,2.5,5}$, for Ansible codebase.</td>
<td>77</td>
</tr>
<tr>
<td>A.2</td>
<td>CEval of $s^j_k$, with $j \in [3-8]$ and $k \in {-0,5,10}$, for Kubernetes codebase.</td>
<td>78</td>
</tr>
<tr>
<td>A.3</td>
<td>CEval of $s^j_k$, with $j \in [3-8]$ and $k \in {-0,10,15}$, for Tensorflow codebase.</td>
<td>79</td>
</tr>
<tr>
<td>A.4</td>
<td>CEval of $s^j_k$, with $j \in [3-8]$ and $k \in {-0,2.5,5}$, for Terraform codebase.</td>
<td>80</td>
</tr>
</tbody>
</table>
## Listings

4.1 Pseudocode of construction of D2F, F2F and Developer To Method Network (D2M) given data from real codebase. ................................................................. 23
4.2 Analyse operation function pseudocode. .................................................. 23
6.1 Timeline generation algorithm pseudocode. .............................................. 36
6.2 Timeline simulation algorithm pseudocode. ............................................. 37
6.3 Simulation model algorithm pseudocode. ................................................ 37
6.4 Commit author selection (selCAuthor) distribution construction pseudocode for third iteration. ................................................................. 43
Acronyms

**AT**  All Time

**BA**  Barabási-Albert Model

**CBAspEval**  Codebase Aspects Evaluation

**CEval**  Complex Evaluation

**CollabPerFile/dist**  Collaboration Done Per File Distribution

**D2F**  Developer To File Network

**D2F/D/dist**  Developer To File Network's Developers Weighted Degree Distribution

**D2F/F/dist**  Developer To File Network's Files Weighted Degree Distribution

**D2M**  Developer To Method Network

**DSA**  Distribution Statistical Analysis

**Dist**  Distribution To Analyze

**Eval**  Evaluation

**F2F**  File to File Network
**F2F/dist**  File To File Network’s Weighted Degree Distribution

**FD2D**  File Developer To Developer Network

**FD2D/dist**  File Developer To Developer Network’s Weighted Degree Distribution

**KS**  Kolmogorov-Smirnov statistic

**MD2D**  Method Developer To Developer

**NS**  Network Science

**RCB**  Real Codebase

**TL**  Time Limited

**VCS**  Version Control Systems

**WD**  Weighted Degree
1 Introduction

Contents

1.1 Introduction ............................................................... 3
1.2 Organization of the Document .......................................... 4
1.1 Introduction

Complex systems are intrinsically difficult to model due to the quantity and heterogeneity of relations between their components. As a consequence of new techniques emerging from different, and seemingly unrelated disciplines, such as algorithms and data structures, statistical physics, dynamical systems and data visualization, a new interdisciplinary field of study, entitled Network Science (NS) [1–10], was established for the study of complex systems. It mainly focuses on real systems modelling and simulation, by considering their evolution in time and the dynamic processes that occur within them. Some remarkable research in this area consider the World Wide Web [11], sexual interactions in humans [12], interactions between proteins in yeast [13], or collaboration networks [14]. Notwithstanding these subject differences, the models of authentic systems have similar properties, namely the power law distributions found in their emergent patterns.

Software can be seen as a social system, which is a category of complex system that models the relations between individuals, groups and institutions forming a coherent whole [15]. A specific point of interest in this social system is the collaboration between developers within a project. Several studies investigate the emergent patterns of this social system [16–18], discovering power law distributions. In this work we aim to comprehend the principles underlying the origins of these patterns, i.e, investigating what are the microscopic mechanisms for the evolution of the collaboration between developers by proposing a model where a set of simple local mechanisms lead to the identified emergent patterns.

To propose a model for these systems, several challenges have to be addressed: (1) how to capture and describe the emergent patterns of developers’ collaboration; (2) what are the microscopic mechanisms that lead to these emergent patterns.

Concerning the first challenge, previous investigations consider teamwork when developers changed the same software module, namely a file [17] or a function [18]. However, none of these works characterizes the developers’ activity, as well as the software modules serving as vehicles of communication, to depict the patterns of the key actors of the collaboration process.

Regarding the second challenge, only Valverde et al. [16] simulate developers’ collaboration aiming to comprehend the mechanisms behind the emergence of the identified emergent patterns. However, this model considers e-mail communication as a method to study collaboration, while according to Zhang et al. [17], changing the source code may be the most important activity of the developers.

Several aspects may hinder the simulation of the collaboration process, as software development is characterized by the existence of teams and file modules, and the patterns behind these structures are in themselves a rather complex object of study [19, 20]. Ultimately, the individual characteristics of a developer may also influence their performance [21], and so theirs and the whole system collaboration. Therefore, these characteristics of software development may need to be described through the local interactions that a developer has with the source code.
Given the inherent described complexity of this problem, complex networks become a key assistant to our investigation, as this approach reduces this system to its single components and relationships \[22\], and provides the methods to study the emergent patterns of the system and the principles underlying its origins.

Our work aims to respond to the following research questions:

- RQ1: What are the emergent patterns of the actors, namely the developers and the software modules, that intervene in the collaboration process?
- RQ2: Is it possible to simulate the emergent patterns of collaboration in real systems? What are the mechanisms that allow their simulation? Are these mechanisms common to different codebases, independent of the followed software development process?

The first question aims to understand the patterns of the interactions between developers and files, whose patterns are defined by the characterization of distributions such as the number of times a developer changes a file and the number of times a file is changed. Revealing such distributions as possible known distributions, e.g. power law or exponential, indicates the scale of the difference of activity done by the different developers, or in different files, respectively. The second research question builds on the first to construct a simulation model for emergent collaboration patterns.

For RQ1, we conclude that, generally, the emergent patterns of the actors involved in the collaboration process are described by power law distributions, across different codebases, independently of their characteristics. Considering the second research question, we find that the emergent patterns of collaboration in real systems can be simulated by simple mechanisms, namely, by developers choosing files according to a preferential attachment rule with initial attractiveness; a power law distribution concerning the number of commits done per developer, and a specific rule to choose which developer should commit next. All of the analysed codebases are simulated by these mechanisms, with slight variations in the last rule and in the initial attractiveness parameter.

### 1.2 Organization of the Document

This thesis is organized as follows:

Chapter 2 discusses software development in Version Control Systems (VCS) and makes a brief introduction on NS, providing sufficient knowledge to comprehend the following chapters.

Chapter 3 reviews pertinent literature to present the state-of-the-art methods used to answer similar topics as the one discussed in this dissertation.

Chapter 4 presents the networks used to simplify the social system of software and how these networks will be analysed to characterize the emergent patterns of the actors involved in the collaboration
process.

Chapter 5 introduces the codebases that will be analysed and answers the first research question. Chapter 6 specify a simulation model for the interactions between developers and files to simulate the emergent patterns observed in Chapter 5. The model is used to find sufficient inputs to reach such emergent patterns, comparing the sufficient inputs for each codebase, and therefore answering the second research question.

Chapter 7 discusses the meaning of our results in Software Engineering and the threats to the validity of our work.

Chapter 8 summarizes the results of this dissertation and provides different paths for future work.
2 Background

Contents

2.1 Software Development in Version Control Systems ....................... 7
2.2 Network Science ....................................................... 7
In this Chapter, we introduce key concepts to understand the following chapters. More specifically, we start by contextualizing software development in VCS and briefly introduce NS.

## 2.1 Software Development in Version Control Systems

One of the main tools to develop software is VCS, such as Github. VCS allow three key properties to develop software: (1) to preserve the history of the work done; (2) to support collaboration while managing conflicts; (3) to branch the code, by creating contained areas, namely to conduct safe experiments. One of the key ideas behind VCS is the concept of commit. A commit is a collection of changes, done by one author, to one or more files on top of a previously defined codebase. Formally, a commit can be defined as a 4-tuple \((id, D, F, d)\), where \(id\) identifies the commit, \(D\) represents the author of the commit, \(F\) represents the set of interactions with files and \(d\) represents the date of the commit. An interaction with a file consists of the changes made to a file, either to its source code or its name. A more fine-grained approach can be taken by considering the interactions with methods instead of files. A commit would be identified similarly, modifying \(F\) to \(M\), which represents the set of interactions with methods. An interaction with a method is defined in the same manner as an interaction with a file, however considering methods instead of files.

VCS have some specificities that should be considered. By allowing branching, the changes of a certain branch can be transferred into another one, which is commonly defined as a merge commit. A merge commit may not add original code, in the case that it simply transfers it. When researching software development through VCS, this should be taken into account.

## 2.2 Network Science

Complex systems have an important role in our daily life, as they surround us across different disciplines. According to Barabási [4], their understanding became one of the major intellectual and scientific challenges of the 21st century. Each complex system can be modelled into an intricate network that simplifies the interactions between the system’s components. Networks facilitate the analysis of this kind of system to a much higher degree than a simple casual inspection. Therefore, to understand complex systems, a deep understanding of the networks behind them needs to be developed.

### 2.2.1 Networks Characterization

A network is a set of the system’s components called nodes and the interactions between them, defined as edges. The manner the edges are defined should dictate the nature of the questions that will be explored, e.g. by linking friends to each other, we obtain a friendship network, from which the analysis
of the transmission of ideas, products and habits is a major interest to sociology, marketing and health sciences [4]. Therefore, different network designs are possible, with the links of a network possibly having directions or weights.

A basic concept of each node is its degree, representing the number of edges connected to it. Considering a network whose edges have directions, instead of degree, the concepts of in-degree and out-degree of a node are used, representing the number of edges directed into the node and the number of edges directed out of the node, respectively.

If the network is weighted and without directions, we can consider the Weighted Degree (WD) of a node, calculated by the number of edges connected to it, but pondered by the weight of each edge, as defined in the following equation:

\[
WD(n) = \sum_{n_i \in N \setminus \{n\}} \text{weight}(n, n_i)
\]

where \(N\) is the set of nodes, \(n, n_i \in N\), and \(\text{weight}(n, n_i)\) represents the weight of the edge between \(n\) and \(n_i\), which is zero if they are not connected.

To characterize the networks’ structure and properties, most properties are calculated with the support of the degree distribution, \(P_k\). It is calculated through the probability that a randomly selected node in the network has degree \(k\). For a network with \(N\) nodes, the degree distribution is the normalized histogram given by:

\[
P_k = \frac{N_k}{N}
\]

where \(N_k\) is the number of nodes with degree \(k\).

2.2.2 Real-world Networks Analysis

As already discussed, NS intends to represent real networks through model building. On the first inquiry, real networks seem to be built on random mechanisms [4]. Given this, one of the first models built was named as Random Network Model and consists of a network with \(N\) nodes where each node pair is connected with probability \(p\). This model fails to model real networks as its degree distribution follows an exponential distribution while real networks follow a distinct distribution that highlights a small number of individuals, commonly defined in the literature as hubs [4].

Hubs are a signature of a principle named scale-free property. A scale-free network is a network whose degree distribution follows a power law, which is characterized by the following Equation, where \(\gamma\) is defined as the power law exponent:

\[
P_k \sim k^{-\gamma}
\]
Normally, to represent the degree distribution in a graph, two techniques are used: (1) the plot has a double logarithmic scale, commonly named a log-log plot, as a linear plot can not display widely different degrees; (2) the cumulative distribution is used, as defined in Equation (2.4), as it allows for clean fits of known distributions, e.g. exponential or power law.

\[ P_{\text{Cum}}^k = \sum_{k' = k}^{\infty} P_{k'} \]  

(2.4)

The main difference between a random and a scale-free network, characterized by exponential and power law distributions respectively, is related to the tail of the distribution, i.e. the high region of \( k \), as observable in Figure 2.1. The distributions in the Figure were generated with support from Python’s Numpy library [23]. The tail of the power law has much more weight, which represents the hubs. In this case, the maximum degree of the power law is 4889 while the exponential has a maximum of 189. The power law can be described by its straight line in the log-log graph, while the exponential is characterized by its cut-off.

![Figure 2.1: Comparison between cumulative distributions of a power law distribution and an exponential distribution.](image)

The distributions can deviate from a perfect power law, showing slight variances [24]. Some examples are: (1) Double power law, which is a power law characterized by two different exponents; (2) Lomax distribution, which behaves like a constant for small values of \( k \), and as a power law for large \( k \); (3) Power law with exponential cut-off, which is a power law for small values of \( k \), but decays exponentially for larger \( x \).

Barabási [4] discusses that in the past decade, different real networks of major scientific, technological and societal importance were studied, revealing the scale-free property. Given this diversity, the scale-free property started to be seen as a universal network characteristic, with some exceptions such
as a power grid network, or networks describing bonds of atoms. The degree distributions of many of these systems deviate from a theoretical power law, due to missing data or data collection biases.

When analysing multiple degree distributions, visual inspection is not enough as this inspection may induce errors, since a power law may appear as a good fit even if the data is drawn from a non-power law distribution [25]. Therefore, to statistically analyse each distribution, a log likelihood-ratio test is conducted considering an exponential distribution and a power law distribution as competing statistical models. This method is based on the work of Alstott et al. [26] which concluded that is generally more useful to compare the fits of many candidate distributions, and identify which one fits best given that: (1) fitting power law distributions is non-trivial; (2) few empirical systems will not follow a theoretical power law distribution, as real systems have noise. In addition to the power law distribution, the exponential distribution is chosen as it is the absolute minimum alternative for evaluating the heavy-tailedness of a distribution [26].

The results of the log likelihood-ratio test are: (1) log-likelihood ratio $R$; (2) significance $p$ of this result, as typically set in statistical hypothesis testing, i.e., 5%. Therefore, when $p > 0.05$, neither distribution is a significantly stronger fit, quoted as AMBIGUOUS\_FIT. Otherwise, the empirical data is more likely to follow: (1) a power law if $R > 0$, expressed as PL\_FIT; (2) an exponential distribution if $R < 0$, defined as EXP\_FIT. If there are no values to do the log-likelihood test, the result of the log likelihood-ratio test is defined as NO\_FIT. In the case of PL\_FIT, the power law can be described through the: (1) $\gamma$ of the fitted distribution according to Equation (2.3); (2) $x_{min}$ which is the optimal minimum $x$ beyond which the scaling regime of the power law fits best. This analysis together with the log-likelihood ratio test will be quoted as Distribution Statistical Analysis (DSA) from now on.

2.2.3 Principles behind Real-world Networks Patterns

To understand how different systems have coincident distributions, some authors investigate how to model the time evolution of the empirically observed topologies. They find that global patterns emerge without a central authority, external bias or collective conscience, i.e. collective phenomena emerged from microscopic interactions.

One of the first models to explain this is proposed by Barabási et al. [27]. Their design consists of two simple properties to explain the universally observed scale-free property: (1) growth, by adding nodes sequentially; (2) preferential attachment, as popularity is attractive, according to Equation (2.5), where $i, j$ represents nodes and $Q(i)$ represent that the probability that a link of a new node connects to $i$ depends on its degree, $k_i$. The main message of this model, commonly named as Barabási-Albert Model (BA), is related to the fact that network structure and evolution are inseparable [4], however, this model does not successfully describe many characteristics of real systems, namely by predicting that $\gamma$ according to Equation (2.3) is equal to 3, while most real networks have $2 < \gamma < 5$. 

10
\[ \prod(i) = \frac{k_i}{\sum_j k_j} \] (2.5)

To explain the distribution of other real networks, some authors propose extensions of this model, leading to the conclusion that the differences in the node’s ability to acquire links affect the network topology [4]. For example, to understand a network that captures the citation patterns of research papers, Eom et al. [28] extend BA to grant an initial attractiveness to every node. In this case, the preferential attachment of the model is characterized by the following Equation:

\[ \prod(i) = \frac{A_i + k_i}{\sum_j A_j + k_j} \] (2.6)

From this equation, it is observable that the BA with attractiveness is similar to BA when \( A_i = 0, \forall i \).

Even if the emergent patterns of two different systems are similar, they may originate from different microscopic mechanisms. Therefore, to test hypotheses on the origins of emergent patterns, large-scale simulations of networked populations need to be run.

### 2.2.4 Impact of Networks Topologies

Network structure should be considered when analysing the dynamical processes of biological, social and technological systems. Their structure has a key role in the respective system’s ability to be robust, resist cascading effects, etc.

Concerning robustness, NS analyses how many nodes we need to delete to fragment the network into isolated components. For the specific case of scale-free networks, if randomly removing nodes, almost all of their nodes must be removed to fragment them. Against targeted attacks, a scale-free network can be deformed by removing an insignificant fraction of all hubs [4].

The previous paragraph assumes that nodes fail independently of each other, however, in reality, the activity of a node depends on the activity of its neighbouring nodes, which means that a node failure can lead to other nodes’ failures. This may lead to cascading failures, which is a threatening event in most networks [29]. It is discovered that their size distribution is well approximated by a power law, meaning that only a small number of cascades have a global impact, and most of them go unnoticed [4].
3 Related Work

Contents

3.1 Relations between Developers ............................................. 13
3.2 Interactions between Developers and Files ............................... 14
3.3 Models that simulate the Development of Software ...................... 15
3.4 Validity of Research ........................................................ 15
3.5 Summary and Discussion .................................................... 16
In this Chapter, we describe research considering the social system behind software development, with the support of methods from NS. We start by highlighting how state-of-the-art research models collaboration between developers into networks and how this collaboration is characterized. Then we specify other designs of networks that may aid us to model other interactions of the system. After this, we describe some works that already formulate models aiming to comprehend the principles underlying discovered emergent patterns. Finally, we detail the validation of the research done.

3.1 Relations between Developers

The social system behind software development is inherently complex. Developers can, for example, collaborate indirectly through changes in files or directly by exchanging emails. There is vast research that explores these relations and improvements in them. Various works have modelled these systems by a network, where nodes represent developers and edges represent a certain type of collaboration.

Bird et al. [30] consider a direct network based on emails, where a link from individual A to individual B denotes that B replied to a message from A. Apache HTTP is used as a case study and it is found that both in and out-degree of this network follows a power law as well as the number of the messages sent and the number of the messages received. Valverde et al. [16] explore the undirected version of this network discovering that the degree distribution roughly follows a power law. However, Zhang et al. [17] discuss source code as the most important activity of the developers.

Regarding data from the source code, various approaches are taken. The idea behind all approaches consists of constructing a network where developers are connected if they change the same software module. Zhang et al. [17] made a systematic analysis of this idea, concluding that these networks can be constructed in different manners, considering, for example, that the edges can be weighted or unweighted since the strengths of developer links may not be always the same. They conclude that these types of networks can fairly capture the collaboration between developers, however, this link between developers is not a straightforward “social” relationship. This relationship is established in source code, not immediate communication.

Regarding unweighted networks, Liu et al. [31] consider a network where a pair of developers are connected if they change at least one file in common, not weighting the edges.

Regarding weighted networks, Lopez et al. [32] create a network where two developers are connected if both contributed to at least a common module. In this case, they weigh the edges by the number of commits performed by both developers to all the common modules, enabling them to differentiate the intensity of collaboration. Roach et al. [33] analyse the Python open-source codebase considering a weighted edge as the number of common files in which two developers have worked. Meneely et al. [34] add a time constraint to these previous studies, by connecting developers that com-
mitted to at least one common file within a month of each other. In their work, an edge is weighted as the number of source code files that the developers did not work on together, which represents the distance between developers. Collaboration is then measured based on the distance between developers.

Joblin et al. [18] consider a partnership when developers change the same function or functions that are semantically coupled. Edges are weighted based on a matrix formula that considers two different matrices: the contributions of developers per function and the coupling between functions. To investigate the project evolution, they inspect a small observation window of 90 days. Also, to avoid discontinuities between the edges of observation windows, they opt for an overlapping-window technique with a step size of half of the window size. This means that if a network starts on January 1st, it will go through April 1st and the following network will start on February 15th, which is 45 days after January 1st.

Notwithstanding these networks differences, it is common to find power law distributions describing emergent patterns of collaboration in developers [18, 31, 32]. Considering unweighted networks, Liu et al. [31] find that their degree distribution follows a power law of different exponents for different systems (1.54 for Angular, 1.75 for Vue.js, 1.38 for Cloud Foundry). Regarding weighted networks, López et al. [32] find power laws through visual inspection. Joblin et al. [18] analyse 17 different systems such as Apache HTTP, Django and Firefox, and conclude that the distribution of power law is not universally present concerning time. In growth networks that surpass 50 developers, a power law distribution for the degrees of the network always emerges. In 12 of these codebases, this property stabilizes and it is rarely lost. In the other codebases, this property is never achieved, remains in an oscillatory state or is lost indefinitely, given the fact that the codebase growth rates decrease and often the number of contributing developers drops.

One exception to the observation of power law distributions is the work of Roach et al. [33] that do not observe one in the Python codebase, with the authors explaining that this might happen because developers are inclined to look in most files, and so, any developer is connected with most other developers.

3.2 Interactions between Developers and Files

The research proposes the design of other types of complex networks to model software development as a complex system. One relevant network in the area, commonly named a contribution network, consists of modelling a bipartite between developers and files, with an edge representing that a developer interacted with a file, by adding, changing or deleting it. The first work to use this network is authored by Pinzger et al. [35], weighting the edges as the number of changes that a developer made to a file. Other works also use this method [36]. In this work, we will reference this network as Developer To File Network (D2F). There is also alternative research using this network considering unweighted edges [33,37].
As far as we know, in the context of collaboration, there is no investigation on the emergent patterns associated with this network, namely the distribution of files changed per developer or the distribution of the number of changes per file.

3.3 Models that simulate the Development of Software

By using preferential attachment and various models of evolving networks, different designs have been proposed to simulate the evolution of software, either the software structure or the collaboration between developers.

To achieve the power law distributions observed in the software structure, a study resorts to the idea of preferential attachment and the concept of fitness [38]. These simple ideas are capable of producing the power laws observed in software systems, in terms of change sizes, class size, number of method calls and subclasses. The results of these simulations are compared with the empirical values of the following systems: jEdit, Eclipse JDT, Apache Maven, and Google Guice.

Other work focus on the relations between packages in the Gentoo Linux operating system [39]. They model a new network with packages as nodes and dependencies among them as edges. They prove that this network cannot be explained by existing complex network models, such as the BA, and create two new evolution models, in which a new node is connected to an old node with a probability that depends on the degree of node but also the age of node. From this work, it is concluded that these new models describe the network better than other existing models.

There is very limited research on the simulation of collaboration between developers. Only Valverde et al. [16] simulate this evolution based on e-mail communication. In their work, they consider that a new member joins the network and sends a small number of emails, per each step of the simulation. The receiver of each new e-mail is determined by a non-local preferential attachment rule, such that the probability of a member receiving an email is proportional to the betweenness centrality of that member in that time step. They conclude that the networks arising from this method comply with the real networks. The real networks are created based on a publicly available electronic database describing e-mail activity containing data from 120 systems from SourceForge, such as Python and TCL. As we know of, there is no research on simulating collaboration between developers given data of source code changes.

3.4 Validity of Research

As stated by Zhang et al. [17], developers establish a relation in source code and not in immediate communication. Since a software module has multiple contexts, developers may not collaborate when
changing the same module, which questions the validity of the research done through networks considering collaboration based on changes in software modules. By conducting a survey, Meneely et al. [34] approach this issue by comparing developers’ perceptions with the results of a network where the nodes are developers and the edges represent that two developers have made a commit to the same file within a one-month interval. Each edge is weighted by the number of files that the pair of developers did not work on together. Joblin et al. [40] address this issue considering collaboration when developers contribute to the same function, weighted by a function of both temporal location and amount of contributed code made through successive changes.

Nevertheless, both works [34, 40] conclude that, in general, their designed developer network is supported by developers’ perceptions. Considering the work done by Meneely et al. [34] in systems such as Linux, PHP programming language and Wireshark, they find that while edges tend to represent collaboration, not every edge represents collaboration and not every collaboration is represented by an edge although empirically the distance between collaborators tends to be less than two. They conclude that the developer network can be treated as an approximation in future applications of social network analysis and that, in general, the developer network is supported by developers’ perceptions. Concerning the work by Joblin et al. [40] they find out that their particular view of collaboration largely coincides with developers’ perceptions of real-world collaboration, in systems such as Linux and QEMU. The predominant source of errors is from missing collaborative links, which means that the links that are identified are largely accurate. In comparison with the network that assumes collaboration if developers contributed to the same source code file, this implies that both cases suffer from missing collaborations, however only the coarse-grained approach (of considering collaboration if developers change the same file) suffers from false collaborations.

Tymchuck et al. [41] conclude that data mined from software repositories are likely to not represent the actual collaboration among developers, as survey results differ from the network results, by analysis of a super-repository called SmalltalkHub. They propose a network that quantifies the collaboration between a pair of developers by computing the number of versions, i.e. commits, that have a preceding version by the other author, considering only versions committed in a time frame of 30 days. In their work, they use a survey with 4 levels of collaboration, criticizing the work of Meneely et al. [34] by using a survey that considers collaboration as a binary option, failing to capture the intensity of collaboration between developers.

### 3.5 Summary and Discussion

Considering the existing work on the characterization of collaboration through software module changes data, we intend to model actual collaboration, by taking into account the diverse best practices already
described. Thus, our interpretation of collaboration will include properties such as edge weighting to
differentiate the intensity of collaboration and time-restricted collaboration to model actual collaboration
between developers. This will allow us to investigate two different open problems: (1) the emergent
patterns of the actors involved in the collaboration process, i.e. developers and files; (2) the simulation of
the collaboration process between developers. Considering the validity of the research, this will remain
a threat to the validity of our work. However, we investigate the possible disadvantage of using files
instead of methods, since files have larger contexts than methods, which may lead to extra erroneous
interpretations of collaboration between developers. For this purpose, we propose a network not only for
collaboration in files but also for collaboration in methods to compare the emergent patterns from both.
If the patterns are similar, we can conclude that using files instead of methods has advantages given the
former’s programming-language independence and suitability for heterogeneous documents [40].
Modelling Approach

Contents

4.1  Approach ...................................................... 19
4.2  Networks .................................................... 19
4.3  Analysis Method ............................................. 25
In this section, we explain how we approach the problem, namely the first research question. We start by describing the general approach to answer it, and then we model the system, by specifying the networks that will support the investigation. After this, we detail some considerations on these networks construction. Finally, we describe how the networks are analysed.

4.1 Approach

To answer RQ1, and aiming to represent the actual interactions between the actors involved in the collaboration process, various networks are proposed, where the WD distribution of either all or just a part of the nodes of the network resembles distributions associated with such actors. By analysing these distributions, the emergent patterns, i.e. the scale of the difference of activity done or received by the different actors, are inferred. The D2F network defined in the Related Work is the first network to be studied as, by considering separately the WD of developers and files, the scale of the difference of activity done by the different developers and in the different files, respectively, can be analysed. Based on D2F, another is constructed, aiming to represent the indirect interactions between developers in files, specifying the developers’ collaboration. By analysing the WD distribution of this second network, it is possible to clarify the scale of the difference in collaboration done by the different developers. Finally, as files are the key vehicle of communication between developers, their relations may have implications for the collaboration between developers. Given this, we consider yet another network representing interactions between files, specifying then files coupling, where the description of the WD distribution reveals the scale of the difference of coupling that each file has in respect to other files. This further supports the characterization of files as they are key actors involved in the collaboration process.

4.2 Networks

The collaboration network has as nodes the developers and as edges the collaboration between developers. An edge between two developers exists if there is a pair of commits, one from each developer, spaced at maximum by a month, that change the same file. To consider the strength of collaboration, the weight between two developers is given by the following Equation:

\[ c_{global}(d_i, d_j) = \sum_{f \in F} c_{count}(f, d_i, sd, f, d_j, sd) \]  

(4.1)

where \(d_i, d_j\) represent two different developers; \(f\) represents a file; \(F\) contains all the files and \(f, d_i, sd\) represents the sequence of dates (sd) of the commits done by \(d_i\) in \(f\). Given both sequences, we calculate, per file, the number of collaborations between the pair of developers with the support
of Equation (4.2). In this Equation, we represent the sequence of commit dates by developer $d_i$ as $sd_i$. This sequence has determined properties such as $size$, the first element of the sequence as $head$ and the elements following the first as $next$. To restrict collaboration in time, we use a function $mod$, which returns the number of months between two time steps. In practical terms, this interpretation of collaboration between two developers sums their collaboration across all files, where the collaboration in a file $f$ is measured by the maximum number of pairs of changes $i, j$ in $f$, one by each developer, where the dates of the changes $i, j$ are separated in maximum by a month. Note that a change can only be part of one pair.

$$c_{count}(sd_i, sd_j) = \begin{cases} 
0 & \text{if } sd_i.size = 0 \lor sd_j.size = 0 \\
c_{count}(sd_i, next, sd_j) & \text{if } mod(sd_i.head, sd_j.head) > 1 \land sd_i.head \leq sd_j.head \\
c_{count}(sd_i, next) & \text{if } mod(sd_i.head, sd_j.head) > 1 \land sd_i.head > sd_j.head \\
1 + c_{count}(sd_i.next, sd_j.next) & \text{if } mod(sd_i.head, sd_j.head) \leq 1
\end{cases}$$

(4.2)

This network is created based on the best practices of previous research in the area: (1) edge weighting, inspired by [32] to differentiate the intensity of collaboration; (2) time constraint, based on [34] to model actual collaboration. From now on, we refer to it as File Developer To Developer Network (FD2D).

The file coupling network has as nodes the files and as edges the coupling between files. The coupling between two files is calculated by the number of commits where the two files occur, and can be represented by the following Equation:

$$CP_{LG}(f_i, f_j) = \#\{c \in C : f_i, f_j \in c.files\}$$

(4.3)

where $f_i, f_j \in F$, $F$ is the set of all files; $C$ is the set of all commits, and $c.files$ is the set of files in commit $c$. From now on, we quote this network as File to File Network (F2F).

To illustrate an example of the construction of these networks, we define a hypothetic commit history in Table 4.1, with the following components: (1) four commits numbered as 1, 2, 3 and 4; (2) two developers named as $D1$ and $D2$; (3) two files named as $F1$ and $F2$. In Figure 4.1, we present the constructed networks given the commit history described in Table 4.1.

<table>
<thead>
<tr>
<th>Commit number</th>
<th>Author</th>
<th>Files on commit</th>
<th>Date (DD/MM/YY)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D1</td>
<td>F1, F2</td>
<td>1/01/2022</td>
</tr>
<tr>
<td>2</td>
<td>D2</td>
<td>F2</td>
<td>2/01/2022</td>
</tr>
<tr>
<td>3</td>
<td>D1</td>
<td>F2</td>
<td>3/01/2022</td>
</tr>
<tr>
<td>4</td>
<td>D2</td>
<td>F2</td>
<td>4/02/2022</td>
</tr>
</tbody>
</table>

As described in Section 2.2.3, the network structure and evolution are inseparable. Therefore, to
evaluate the patterns over time, each network, which can be defined as an All Time (AT) network, is branched into a set of sub-networks. Similar to the work of Joblin et al [18] which creates a set of overlapping time windows, the \( n^{th} \) sub-network will contain a set \( W_n \) of commits, such that
\[
W_n = \{ c_t : t \in [t_0 + n \times \Delta_{\text{step}}, t_0 + n \times t_{\text{step}} + \Delta_{\text{window}}] \},
\]
where \( c_t \) is the commit occurring at time \( t \), \( t_0 \) is the time of the initial commit, \( \Delta_{\text{window}} \) is the window size, and \( \Delta_{\text{step}} \) is the step size. As in the work of Joblin et al [18], we use a small enough observation window of 90 days, with a step size of half the window. This set of sub-networks will be defined as Time Limited (TL) networks.

Additionally, similar networks to D2F and FD2D can be built where instead of files we consider methods, having then Developer To Method Network (D2M) and Method Developer To Developer (MD2D). The latter network will be useful for Chapter 7.

### 4.2.1 Network Construction

In D2F, part of the nodes represent files and in F2F all nodes represent files, identified by their names. Considering a file, it was previously discussed in Section 2.1 that its name can change. As the name before and after the commit of each file is explicitly defined, the data mined from the VCS should contain this information, per file changed. By taking this process into account and relabeling the file nodes as the files change their names, we track the collaboration of developers per file, even in the case that a file change name.

In Section 2.1 we discussed that methods can change name, which is relevant for D2M construction. However, considering a method, the pair of its names, before and after the commit, are not explicitly defined and analysis of the source code should be conducted to determine both names. This analysis is not deterministic as the method name and method source code can be defined independently by the author, leading to errors. An example of a non-deterministic case can be seen in Figure 4.2. For this specific case, it is arguable if the method named \( b \) is either renamed from method \( a \) or a new method, in
which case method \texttt{a} is deleted.

\begin{verbatim}
  function a()
  
  print(1)
  
  }

(a) File f source code before commit.

(b) File f source code after commit.
\end{verbatim}

\textbf{Figure 4.2:} Example of a non-deterministic method rename.

Given the problem non-determinism, in our work, methods are never relabeled, i.e. considering the set of methods before the commit \(mb\), and the set of methods after the commit \(ma\), if a given method \(m\) exists only in one of these sets, then: (1) \(m\) is deleted if it is in \(mb\); (2) \(m\) is a new method if it is in \(ma\). Considering the example given in Figure 4.2, method \(a\) is deleted and method \(b\) is added. Note also that a method identification always requires the file he belongs to, as different files may have different methods with the same name.

In Listing 4.1, a pseudo code illustrative of D2F, F2F and D2M construction is depicted, defining:

- \texttt{addNode(name, type)} as a function which adds a node named as \textit{name} with type \textit{type}.

- \texttt{concatenate(s1, controlString, s2)} as a function which concatenates two strings \textit{s1} and \textit{s2}, separated by a string named as \textit{controlString}. The control string can be any string that does not exist in file names.

- \texttt{addEdgeOrWeight(n1, n2)} as a function which adds weight to the edge between nodes \textit{n1} and \textit{n2} if it exists else create such edge

The code for \texttt{analyseOperation} function, which saves an operation into the respective network is shown in Listing 4.2, defining:

- \texttt{relabel(previousNodeName, newNodeName)} as a function which relabels the node with name \textit{previousNodeName} to \textit{newNodeName}

- \texttt{getNode(n)} as a function from D2F that gets information about the node \textit{n}

- \texttt{getMethods()} as a function which gets the methods that exist in a file

- \texttt{add(item)} as a function from sequence that inserts an item \textit{item} into the end of the sequence

- \texttt{changeStatus(newStatus)} as a function which changes the status of a node
Listing 4.1: Pseudocode of construction of D2F, F2F and D2M given data from real codebase.

```plaintext
Listing 4.2: Analyse operation function pseudocode.
```

Listing 4.1: Pseudocode of construction of D2F, F2F and D2M given data from real codebase.

```plaintext
Listing 4.2: Analyse operation function pseudocode.
```
begin
  if itemOp.op == ADD {
    d2i.addNode (itemOp.itemNameAfter, type)
    d2i.addEdgeOrWeight (author, itemOp.itemNameAfter)
  }
  if type == FILE {
    commitFiles.add (itemOp.itemNameAfter)
  }
}

if itemOp.op == CHANGE {
  if type == FILE {
    if itemOp.itemNameBefore != itemOp.itemNameAfter {
      d2i.rename (itemOp.itemNameBefore, itemOp.itemNameAfter)
      d2f.rename (itemOp.itemNameBefore, itemOp.itemNameAfter)
      if the file is relabeled, as the method name also refers to a file, it also needs relabeling
        fileMethods = d2i.getNode (itemOp.itemNameBefore()).getMethods()
        for i := 1..len(fileMethods) {
            m = fileMethods[i]
            changeMethodNameOp = new ItemOperation (op = CHANGE
                itemOp.itemNameBefore = concatenate (itemOp.itemNameBefore, controlString, m)
                itemOp.itemNameAfter = concatenate (itemOp.itemNameAfter, controlString, m)
            )
            analyseOperation (author, d2m, None, changeMethodNameOp, None, type = METHOD, canRelabelMethod=True)
        }
      commitFiles.add (itemOp.itemNameAfter)
    }
  } else if type == METHOD {
    if itemOp.itemNameBefore != itemOp.itemNameAfter {
      if canRelabelMethod { // can only relabel when file name is changed
        d2i.rename (itemOp.itemNameBefore, itemOp.itemNameAfter)
      } else {
        // delete old method
        d2i.addEdgeOrWeight (author, itemOp.itemNameBefore)
        d2i.changeStatusOfNode (itemOp.itemNameBefore, DELETED)
      // create new method
        d2i.addNode (itemOp.itemNameAfter)
        d2i.addEdgeOrWeight (author, itemOp.itemNameAfter)
      }
    }
  }
}

if itemOp.op == DELETE {
  d2i.addEdgeOrWeight (author, itemOp.itemNameBefore)
  d2i.getNode (itemOp.itemNameBefore).changeStatus (DELETED)
  if type == FILE {
    commitFiles.add (itemOp.itemNameBefore)
  }
}
return d2i, d2f, commitFiles
end
4.3 Analysis Method

Considering that the WD of a node represents the number of interactions that this node has with other nodes, the analysis of the WD distribution describes the scale of the difference of interactions done by the different nodes of the network, i.e. the emergent patterns of such network. More specifically, to answer RQ1, the distributions of the WDs of D2F and F2F will be analysed for several open-source codebases, as they represent the emergent patterns of the interactions existing between developers and files, or just files, respectively. In particular, developers are studied through Developer To File Network’s Developers Weighted Degree Distribution (D2F/D/dist), which represents the distribution of the number of changes in files per developer. Regarding files, the analyses are: (1) Developer To File Network’s Files Weighted Degree Distribution (D2F/F/dist) depicting the distribution of the number of times a file is changed; (2) File To File Network’s Weighted Degree Distribution (F2F/dist) representing the distribution of the sum of the number of files in commits where a certain file is present. Completing the answer to RQ1, as our interpretation of collaboration was never studied given that it consists of the collection of the best practices used in previous works, we also study File Developer To Developer Network’s Weighted Degree Distribution (FD2D/dist), which represents the distribution of the number of times that a developer has collaborated with other developers, in files. The description of the various distributions is supported by: (1) visual inspection of their representation; (2) DSA described in Chapter 2.
Empirical Emergent Patterns

Contents

5.1 Data Gathering ................................................................. 27
5.2 Collaboration Actors Patterns .............................................. 27
5.3 Collaboration Patterns ....................................................... 30
In this Chapter, we present the answer for RQ1. We start by introducing the investigated codebases alongside their characteristics and the method used to gather the data. After this, we describe the various distributions associated with the actors intervening in the collaboration.

5.1 Data Gathering

We focus on gathering data from a set of GitHub’s codebases, namely Ansible, Bitcoin, Kubernetes, Tensorflow and Terraform. These codebases are chosen due to their degree of complexity, by having more than twenty thousand commits done and at least one thousand developers that made at least one commit. The gathered data concerns the history of all commits. In Table 5.1 we present relevant information about them.

Table 5.1: Characteristics of investigated codebases.

<table>
<thead>
<tr>
<th>Codebase</th>
<th>Total commits</th>
<th>Total developers</th>
<th>Total files</th>
<th>Creation year</th>
<th>Most used programming language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ansible</td>
<td>46521</td>
<td>5834</td>
<td>25387</td>
<td>2012</td>
<td>Python (88.4%)</td>
</tr>
<tr>
<td>Bitcoin</td>
<td>22875</td>
<td>1001</td>
<td>3652</td>
<td>2009</td>
<td>C++ (67.4%)</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>61838</td>
<td>3680</td>
<td>38470</td>
<td>2014</td>
<td>GO (96.7%)</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>114742</td>
<td>3430</td>
<td>38433</td>
<td>2015</td>
<td>C++ (62.7%)</td>
</tr>
<tr>
<td>Terraform</td>
<td>23603</td>
<td>1810</td>
<td>14467</td>
<td>2014</td>
<td>GO (99.4%)</td>
</tr>
</tbody>
</table>

To facilitate this extraction, we use Pydriller [42] which is a Python framework that extracts information about commits, developers, modified files, etc. Data is taken out of the main branch of every repository. Every commit is simplified into a set of relevant attributes to our study: commit author, date, files changed in the commit and the methods changed per file. For files, we consider the name of the file before the commit and after, as discussed in Section 4.2.1. Concerning methods, the renames are not explicitly defined, having, per file, the names of the methods before the commit and after the commit. Given both of these sets, if a method exists only in one of them, it is considered either as a delete or an add, respectively, as discussed in Section 4.2.1.

5.2 Collaboration Actors Patterns

Considering developers, the representation and DSA of D2F/D/dist for the distinct codebases, with respect to AT D2F networks, is shown in Figure 5.1. For any codebase, the distributions are better fitted by a power law than an exponential. Except for Kubernetes, \( \gamma \) has similarities across all codebases. Through visual inspection, we observe that Kubernetes distribution follows a double power law, which is according to the higher \( x_{min} \), with Kubernetes \( \gamma \) representing the exponent of the rightest power law.
In Table 5.2 we describe the percentage of each D2F/D/dist DSA result across the different TL D2Fs per codebase. By observing the table we can conclude that: (1) Power law distributions are virtually universally present in the Ansible codebase; (2) D2F/D/dist from Tensorflow and Terraform TL D2F are mainly described by power law fits having a small percentage of TL D2F whose distribution can not be better described by a power law distribution nor an exponential one; (3) considering Bitcoin and Kubernetes, the majority of TL D2F distributions are not better described by a power law nor an exponential. Given these results, a more thorough analysis of these distributions needs to be proposed, as we can not conclude if the power law property is universal concerning time. We leave this as an open problem as this is not the main intent of our work.

Table 5.2: Percentage of D2F/D/dist DSA results for the distinct codebases regarding TL D2Fs.

<table>
<thead>
<tr>
<th>Codebase</th>
<th>% AMBIGUOUS_FIT</th>
<th>% PL_FIT</th>
<th>% EXP_FIT</th>
<th>% NO_FIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ansible</td>
<td>2</td>
<td>98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bitcoin</td>
<td>39</td>
<td>55</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>63</td>
<td>37</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>28</td>
<td>72</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Terraform</td>
<td>18</td>
<td>82</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

With respect to files, the DSA of D2F/F/dist for the distinct codebases regarding AT D2F networks, and its visual representation can be seen in Figure 5.2. Aside from Terraform, power laws are a better description in comparison with exponential distributions, as \( R > 0 \) and \( p < 0.05 \). By inspecting Terraform, we observe the existence of a heavy tail with some files being highlighted in relation to a possible power law fit. In this case, all \( \gamma \) are similar.

The percentage of each D2F/F/dist DSA result across the different TL D2Fs is described in Table 5.3. In each codebase, the majority of the distributions are described by a power law in favour of an exponential, however, it is not possible to conclude the universal presence of a power law concerning time.
as there is a relatively high percentage of distributions that are not better fitted by a power law nor an exponential. Therefore, as in D2F/D/dist, a thorough analysis of these distributions is necessary, leaving this as an open problem.

**Figure 5.2:** Representation and DSA of D2F/F/dist for the distinct codebases regarding AT D2F networks.

Table 5.3: Percentage of D2F/F/dist DSA results for the distinct codebases regarding TL D2Fs.

<table>
<thead>
<tr>
<th>Codebase</th>
<th>% AMBIGUOUS_FIT</th>
<th>% PL_FIT</th>
<th>% EXP_FIT</th>
<th>% NO_FIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ansible</td>
<td>32</td>
<td>68</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bitcoin</td>
<td>40</td>
<td>60</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>22</td>
<td>78</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>28</td>
<td>72</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Terraform</td>
<td>18</td>
<td>82</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

With respect to F2F/dist, the representation and DSA of this distribution for the AT F2F networks is shown in Figure 5.3. By visual inspection, the distributions are similar to a Lomax distribution. It means that exists only a small number of files that are committed with a small number of other files. This can be explained by the fact that most files are coupled with other files. Statistically, except for Ansible and Bitcoin, a power law distribution is a better fit in comparison with an exponential, for the remaining codebases. For Ansible, it is possible to observe a power law with an exponential cut-off and some files highlighted concerning a possible fit of a power law. Visually, Bitcoin distribution has a heavy tail, by comparison with the other distributions. Considering $\gamma$, it varies slightly between 2.37 and 3.3 for the distinct codebases.

With respect to the F2F/dist across the various TL F2Fs, the percentage of the DSA results are exhibited in Table 5.4. We can conclude that for the different codebases, the majority of F2F/dist are neither better fitted by a power law nor an exponential. Note also that some of the F2F/dist of TL F2F are better described by an exponential than a power law. As in the previous analysis done for D2F/D/dist and D2F/F/dist concerning TL networks, a more specific statistical analysis of these distributions is required,
and we leave this as an open problem.

Table 5.4: Percentage of F2F/dist DSA results for the distinct codebases regarding TL F2Fs.

<table>
<thead>
<tr>
<th>Codebase</th>
<th>% AMBIGUOUS_FIT</th>
<th>% PL_FIT</th>
<th>% EXP_FIT</th>
<th>% NO_FIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ansible</td>
<td>58</td>
<td>27</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Bitcoin</td>
<td>73</td>
<td>16</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>48</td>
<td>52</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>57</td>
<td>41</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Terraform</td>
<td>57</td>
<td>26</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

5.3 Collaboration Patterns

For analysis of the collaboration emergent patterns, the representation and DSA of FD2D/dist for the AT FD2D networks across the different codebases can be observed in Figure 5.4. For the various codebases, a power law is a better fit in comparison with an exponential, which is in agreement with the observations done by Lopez et al. [22]. Besides Kubernetes, different codebases have similar $\gamma$. Considering Kubernetes, its distribution follows a double power law, with its $\gamma$ representing the exponent of the rightest power law.

In Table 5.5 the percentages of the results for the FD2D/dist across the different TL FD2Ds are presented. We can observe two patterns: (1) FD2D/dist from Ansible and Terraform TL FD2Ds are mainly described by power law fits having a small percentage which is not better described by a power law distribution nor an exponential one; (2) the majority of FD2D/dist of TL FD2Ds for the rest of the codebases are not better described by a power law nor an exponential. Given this, as discussed for the previous distributions, a more thorough analysis of these distributions needs to be proposed, as we can not conclude if the power law property is universal or not. We leave this as an open problem as this is
Table 5.5: Percentage of FD2D/dist DSA results for the distinct codebases regarding TL FD2Ds.

<table>
<thead>
<tr>
<th>Codebase</th>
<th>% AMBIGUOUS</th>
<th>% PL</th>
<th>% EXP</th>
<th>% NO</th>
<th>% FIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ansible</td>
<td>18</td>
<td>82</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bitcoin</td>
<td>87</td>
<td>8</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>72</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>80</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Terraform</td>
<td>29</td>
<td>72</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Answering the first research question, the emergent patterns are described by power law distributions, across different codebases, having similar power law exponents. Therefore, for the various distributions, there is a small number of actors with a high number of interactions, in relation to the rest of the actors who make a small number of interactions. Considering the evolution of these patterns across time, a thorough analysis of the various TL networks distributions should be done, as generally, there exist distributions that can be neither better fitted by a power law nor an exponential. We leave the investigation of the evolution of the distributions over time as an open problem as it is not the main focus of our work.
6

Simulation of the Emergent Patterns

Contents

6.1 Approach .................................................. 33
6.2 Algorithm .................................................. 35
6.3 Results .................................................. 37
In this section, we define a parametric simulation model that tries to reproduce the evolution of the collaboration between developers according to the characterization done in the first research question. By doing so, we can identify sufficient principles capable of replicating the empirical observations.

This simulation is done iteratively, initially considering the most straightforward aspects, and augmented with additional layers of complexity for the following iterations until the results are similar to the real systems. Finally, we compare the results achieved from the simulation model for different existing systems, showing how the impact of these principles can be qualitatively similar across codebases, and the differences in the applicability of such mechanisms in the various contexts.

6.1 Approach

To answer RQ2, we create a general algorithm that simulates the interactions between developers and files in a codebase. In particular, this algorithm receives as parameters: (1) a set of codebase attributes \( ca \); (2) a set of distributions \( \text{dists} \) defining how these attributes should be assigned for their recipients; (3) rules indicating how developers choose files to change (\( \text{chR} \)) and delete (\( \text{delR} \)). The study will be done through variations of these parameters.

For simplicity, we shall depart from a particular Real Codebase (RCB), and given the vast quantity of variables to create and simulate a timeline of commits, we assume that the parameter (1) of the algorithm to be equal to RCB attributes.

As said, we aim to discover what are the sufficient mechanisms to achieve the emergent patterns of different RCBs. For this purpose, considering a RCB, parameter (2) starts with the most simple parameters that will be changed given the results of the algorithm. For parameter (3), a set of rules is fixed, not only to study different possible hypotheses but also for a chance method. This method serves as a term of comparison to check if the discovered emergent patterns from the various rules display some properties that would not be likely based on chance alone, i.e. non-trivial properties, namely, power law distributions.

More specifically, each evaluation of the parameter (2) considers the results of the simulation, i.e. networks \( \text{D2F}, \text{FD2D} \) and \( \text{F2F} \), by comparing, for each network, the WD distribution of the network with the respective empirical distribution, i.e. the respective distribution of RCB, as the latter resemble the target emergent patterns. This comparison is done through three approaches: (1) conduct a DSA and verify if the best fit distribution is equal to the empirical one; 2) conduct a goodness of fit test with Kolmogorov-Smirnov statistic (\( \text{KS} \)) [25]; (3) if a power law is the best fit for both distributions, compare both exponents.

To construct a timeline similar to the one of a determined RCB, a set of aspects of a codebase are compared between the results of the algorithm and RCB, through KS. We define this analysis as
Codebase Aspects Evaluation (CBAspEval). This set of aspects is described until the specification of the algorithm. For this description, consider $t$ as a time step and that the $i^{th}$ commit is related to the $i^{th}$ time step of the timeline of commits.

Before revealing the aspects, we define metrics to measure the uniformity of commits over time per developer, which characterizes the frequency of activity of developers. Uniformity of commits should be considered either in: (1) the codebase’s entire existence; (2) the interval between the first and last commit per developer. For (1), we define the following metric:

$$U_{global}(d) = \sum_{i=1}^{c_d} \text{abs}(t_i - i \times \frac{noc}{c_d + 1})$$

where $d$ represents a developer; $t_i$ represents $d$’s $i^{th}$ commit $t$ since the first codebase commit; $noc$ represents the codebase number of commits; $c_d$ represents the number of commits done by $d$. In practice, this metric measures the difference between the actual time of their commits with a time-uniform distribution of their commits. This metric can be refined for (2), as $U_{local}$, with $t_i$ representing $d$’s $i^{th}$ commit $t$ since $d$’s first commit and $noc$ representing the number of commits between $d$’s first and last commit. Note that only developers with three or more commits can be measured in terms of $U_{local}$.

Both metrics $U_{global}$ and $U_{local}$ are normalized into $\tilde{U}_{global}$ and $\tilde{U}_{local}$, to compare them across developers, as defined in the following Equation:

$$\tilde{U}(d) = \frac{U(d)}{\max(U)}$$

where $c_{d'} = c_d$, however $d$’s commits are done in the first $c_{d'}$ interval time steps implying the highest possible value for $U$ regarding developers with $c_d$ commits.

Regarding developers, the studied aspects of a CBAspEval are: (1) distribution of number of commits per developer ($\text{commitsPerDevDist}$); (2) number of different developers until each $t$ ($\text{newDevelopersUntilT}$); (3) $\tilde{U}_{global}$ distribution ($\tilde{U}_{global\text{dist}}$); (4) $\tilde{U}_{local}$ distribution ($\tilde{U}_{local\text{dist}}$). With respect to files, the number of total adds, changes, and deletes until each $t$ are investigated, defining $\text{filesAddedUntilT}$, $\text{filesChangedUntilT}$ and $\text{filesDeletedUntilT}$. Finally, the time interval between each $t$ and the first time step is also inspected, defining $\text{timePassedUntilT}$. To allow the use of KS, the not normalized aspects are normalized by dividing: (1) each value of $\text{newDevelopersUntilT}$ by the empirical number of developers; (2) each value of $\text{filesAddedUntilT}$, $\text{filesChangedUntilT}$ and $\text{filesDeletedUntilT}$ by the empirical number of files added/changed/deleted, respectively; (3) each value of $\text{timePassedUntilT}$ by the total time of the empirical codebase existence.
6.2 Algorithm

The algorithm has two parts. In the first part, a timeline of commits is created, with each commit being characterized by the following components: author; date; number of files added; number of files changed; and number of files deleted. This first part is divided into two phases: (1) generate, using a distribution, the values for the different commit attributes, independently; (2) aggregate the results of the first phase into each of the commits in the timeline sequence. A pseudo-code illustrating this part can be seen in Listing 6.1, with the following definitions:

- **InputDistributions** is an object that contains distributions for each component to be created. The algorithm accepts known distributions such as uniform or power law, however, it can be evolved to accept any distribution.

- **generate(size, sum, dist, p)** creates a sequence of values with size, sum and distribution dist. The last parameter, p, should only be considered for the components related to the number of files added, changed, and deleted as p represents the percentages of commits with more than 0 files added, changed or deleted, respectively, with the resulting sequence also complying with this restriction.

- **selectCommitAuthor(comPerAuthor, dist)** chooses a commit author using the commits per author in comPerAuthor and a distribution defined as dist.

The second part of the model simulates the process of development by time steps, according to the timeline generated in the first part of the model. In Listing 6.2 we present a pseudo-code illustrative of the second part, defining:

- **d2f, f2f, fd2d** as networks that will be constructed with the same reasoning than D2F, F2F and FD2D, respectively.

- **updateAuthorFile(author, file, interactionType)** permits the increase of the weight of the edge between developer author and file or its creation in case it does not exist; when interactionType = deleted it updates the file status to not receive more commits.

- **updateFileFile(file1, file2)** increases the weight of the edge between two files or creates it if it does not exist.

- **selectFileToDelete(files, delR)** chooses a file to delete from files, through a distribution created based on a rule defined as delR.

- **selectFileToChange(files, chR)** chooses a file to change from files, through a distribution created based on a rule defined as chR.
• `createEdge(d1, d2, weight)` creates an edge between developers `d1` and `d2` with weight equal to `weight`.

• `cglobal(d2f, d1, d2)` calculates collaboration between developers `d1` and `d2` according to Equation (4.1) and the interactions saved in `d2f`.

The rules stated in the input of the second part, `chR` and `delR`, can be rather simple, such as choosing a random file, leading to the creation of a uniform distribution or having more sophisticated methods. In Listing 6.3 we present the pseudo code considering the whole algorithm of the simulation model.

```
Listing 6.1: Timeline generation algorithm pseudocode.

1 CodebaseAttributes {
2   // numberOfDevelopers
3   int nod // number of developers
4   int noc // number of commits
5   int cbDur // codebase duration
6   int fAdd // total files added
7   int pcAdd // percentage of commits with more than zero files added
8   int fCh // total files changed
9   int pcCh // percentage of commits with more than zero files changed
10  int fDel // total files deleted
11  int pcDel // percentage of commits with more than zero files deleted
12 }

13 InputDistributions {
14   Distribution nocPerDev // number of commits per dev dist
15   Distribution dPerC // dates of commits dist
16   Distribution fAddPerC // files added per commit dist
17   Distribution fChPerC // files changed per commit dist
18   Distribution fDelPerC // files deleted per commit dist
19   Distribution selCAuthor // select commit author dist
20 }

21 inputs : Codebase Attributes : ca, InputDistributions : d
22 outputs : sequence
23 begin
24   int comPerAuthor[ca.nod] = generate(ca.nod, ca.noc, d.nocPerDev, None)
25   int datesPerCommit[ca.noc] = generate(ca.noc, ca.cbDur, d.dPerC, None)
26   int addsPerCommit[ca.noc] = generate(ca.noc, ca.fAdd, d.fAddPerC, ca.pcAdd)
27   int changesPerCommit[ca.noc] = generate(ca.noc, ca.fCh, d.fChPerC, ca.pcCh)
28   int deletesPerCommit[ca.noc] = generate(ca.noc, ca.fDel, d.fDelPerC, ca.pcDel)
29   for i = 1..noc+1 {
30     author = selectCommitAuthor(comPerAuthor, d.selCAuthor)
31     comPerAuthor[author] --
32     commit = new Commit(author, datesPerCommit[i], addsPerCommit[i], changesPerCommit[i], deletesPerCommit[i])
33     timeline.add(commit)
34   }
35   return timeline.sortByDate()
36 end

36
```
Listing 6.2: Timeline simulation algorithm pseudocode.

```plaintext
1 inputs:
2 CodebaseAttributes : ca,
3 sequence[Commit] : timeline,
4 string : chR, delR
5 outputs:
6 d2f, f2f, fd2d : network
7 begin
8 for i = 1..ca.noc {
9     commitFiles = []
10     author := timeline[i].author
11     for j = 1..timeline[i].numberOfDeletes:
12         file = selectFileToDelete(d2f.files, delR)
13         d2f.updateAuthorFile(author, file, deleted)
14         commitFiles.add(file)
15     for j = 1..timeline[i].numberOfChanges:
16         file = selectFileToChange(d2f.files, chR)
17         d2f.updateAuthorFile(author, file, changed)
18         commitFiles.add(file)
19     for j = 1..timeline[i].numberOfAdds:
20         file = d2f.createFile()
21         d2f.updateAuthorFile(author, file, added)
22         commitFiles.add(file)
23     for j = 1..len(commitFiles):
24         for k in j+1..len(commitFiles):
25             f2f.updateFileFile(commitFiles[j], commitFiles[k])
26     for i = 1..len(d2f.developers):
27         for j = i+1..len(d2f.developers):
28             d1 = d2f.developers[i]
29             d2 = d2f.developers[j]
30             fd2d.createEdge(d1, d2, cglobal(d2f, d1, d2))
31     return d2f, f2f, fd2d
32 end
```

Listing 6.3: Simulation model algorithm pseudocode.

```plaintext
1 inputs: CodebaseAttributes : ca, InputDistributions : dists, string : chR, delR
2 outputs: d2f, f2f, fd2d
3 timeline = generateTimeline(ca, dists)
4 return simulateTimeline(timeline, chR, delR)
```

6.3 Results

Formally, we define a simulation $s^t_k$ as the inputs of the algorithm described in Listing 6.3, i.e. a tuple $(sCa, sDists_j, h_k)$, representing the inputs for parameters $ca$, $dists$ and $chR$, respectively. Input for $delR$ is omitted as it is found that deleting a file randomly is sufficient to achieve the target emergent patterns. As described before, considering simulations associated with a determined RCB: (1) $sCa$ is constructed based on the real values of a RCB; (2) $sDists_j$ is initiated with the most simple parameters.
and evolved according to the results of the algorithm. In the case of (2), each evolution corresponds to a new iteration, with \( j \) equal to the number of the iteration. Note that: (1) simulations for the same iteration use similar timelines, i.e. for \( s_{jk} \), with fixed \( j \) and varying \( k \), the generated timelines are identical; (2) the algorithm is run five times for each \( s_{jk} \), with its results being the average of the algorithm results.

As discussed in Chapter 3, Barabási et al. [27] found that power law distributions can be simulated by preferential attachment rules. As the RCB distributions found through RQ1 are similar to power laws, we are inspired by the works of Eom et al. [28], which uses a preferential attachment rule with initial attractiveness. Translating this preferential rule into the context of our work, the probability of choosing file \( i \) is characterized by the Equation (2.6), where \( A_i \) represents the initial attractiveness of file \( i \) and \( k_i \) represents the number of changes that file \( i \) received.

Therefore, \( h_k \) can be described by: (1) choosing a random file, i.e. the chance method, defined as \( h_\cdot \) or null hypothesis; (2) our set of hypothesis rules based on Equation (2.6), with each hypothesis having different \( A \). In each of these hypotheses, \( A_i = A_j, \forall i, j \in F \) where \( F \) is the set of all files, i.e. \( A \) is similar to all files. \( k \) represents the \( A \) parameter used in such a hypothesis.

In each iteration, \( h_\cdot \) and other three rules that have more similarities with the empirical results are thoroughly analysed. The reason behind the specific count of three is related to the possibilities of results from the creation of a simulation based on Equation (2.6). The smaller \( A \) is, the bigger the tail of the distribution will be. Given this, considering an iteration \( j \) and assuming that exists an interval \( A = [k1, k2] \) such that \( k \in A \) and the distribution resulting from simulation \( s_{jk} \) is close to the empirical distribution, then, there are two intervals: (1) \( B = [0, k1] \), where the distribution resulting from \( s_{jb} \), with \( b \in B \), has a heavier tail than the empirical distribution; (2) \( C = ]k2, \infty[ \), where the empirical distribution has a heavier tail than the distribution resulting from \( s_{jc} \), with \( c \in C \). Therefore, to exemplify this, we choose three possibilities for \( k \): \( \{k1, k2, k3 : k1 < k2 < k3\} \), where \( k1 \in B, k2 \in A, \) and \( k3 \in C \). For our simulations, \( k1 \) is always equal to 0, given that when \( A_i = 0, \forall i \) for Equation (2.6), the rule becomes similar to the preferential attachment rule for the BA model, as described in Chapter 2.

Considering the final results of the model, i.e. D2F, FD2D and F2F, we define Distribution To Analyze (Dist) \( \in \{D2F/dist, F2F/dist, FD2D/dist\} \). Each simulation is evaluated across the possible Dist, then, we define an Evaluation (Eval) as a tuple \( (s_{jk}^\ast, \text{Dist}) \) where Dist from simulation \( s_{jk}^\ast \) is analysed. Per iteration, the evaluations will be grouped by Dist, to compare the variations of \( h_k \). The results of Eval are:

- \( R_t, p_t, \gamma_t \), results of DSA for the resulting Dist of \( s_{jk}^\ast \). \( \gamma_t \) will be compared with \( \gamma_e \), representative of the power law exponent of the empirical Dist. In the case that \( \gamma_t = - \), it signifies a lack of heavy-tailedness in the distribution.
- KS between the resulting Dist of \( s_{jk}^\ast \) and the Dist from RCB.

The development of simulation iterations will be thoroughly described for the Bitcoin codebase, serv-
ing as an example of the method taken. The simulation iterations for the previously analysed codebases can be observed in Appendix A.

6.3.1 First Iteration

The first iteration is characterized by having uniform distributions for the different attributes in the object InputDistributions, defined in Listing 6.1. This means that all developers make the same number of commits, all commits where files are added/changed/deleted have the same number of added/changed/deleted files respectively, and all of the consecutive commits are separated by the same time interval. Besides this, attribute selCAuthor is equal to a uniform distribution, which means that a random choice is made. This set of distributions will be quoted from now on as sDists$_1$.

Considering RCB = Bitcoin codebase, our set of rules are $h_-$, $h_0$, $h_{10}$, $h_{15}$, meaning that simulations of the first iteration are $s^k_1$, with $k \in \{-,0,10,15\}$. In Figure 6.1 we represent, per different Dist, the results of $s^k_1$ in comparison with the empirical results alongside the respective Evals.

Considering Dist = D2F/F/dist, all simulations except $s^k_1$ achieve the non-trivial properties displayed by the empirical distribution.

For Dist = F2F/dist, in terms of non-trivial properties, we have similar conclusions to the ones found for Dist = D2F/F/dist. However, as KS $\geq 0.4$, there is some difference between theoretical and empirical. Through visual inspection, we observe that the KS is high given that the results of the various simulations do not have the property of the Lomax distribution for small values observed in the empirical distribution.

As our focus is the developers’ collaboration, iterations are developed according to the results of Dist = FD2D/dist.

Finally, in terms of collaboration, i.e. Dist = FD2D/dist, no simulation achieves the non-trivial properties of the empirical distribution. This may be explained by the fact that $sDists$_1nocPerDev is equal to a uniform distribution while in reality, the distribution of the number of commits per developer is similar to a power law. This prompts us to a new iteration.

6.3.2 Second Iteration

$sDists$_2 is developed, in a similar way to $sDists$_1, changing $sDists$_2nocPerDev to a power law. Per different Dist, the results of simulations $s^k_2$, with $k \in \{-,0,10,15\}$ are represented in Figure 6.2 in comparison with the empirical Dist, alongside the respective Evals.

With respect to Dist $\in \{D2F/F/dist, F2F/dist\}$ the results are similar to the ones described in the first iteration.

Considering Dist = FD2D/dist, all the simulations achieve the non-trivial property of the empirical distribution, however, all of them have different properties of the empirical one, as KS $> 0.52$ and $\gamma_t \sim$
(a) Representation of D2F/F/dist resulting from simulations $s^1_k$, with $k \in \{-, 0, 10, 15\}$ in comparison with empirical D2F/F/dist.

(b) Evals ($s^1_k$, D2F/F/dist), for $k \in \{-, 0, 10, 15\}$.

(c) Representation of F2F/dist resulting from simulations $s^1_k$, with $k \in \{-, 0, 10, 15\}$ in comparison with empirical F2F/dist.

(d) Evals ($s^1_k$, F2F/dist), for $k \in \{-, 0, 10, 15\}$.

(e) Representation of FD2D/dist resulting from simulations $s^1_k$, with $k \in \{-, 0, 10, 15\}$ in comparison with empirical FD2D/dist.

(f) Evals ($s^1_k$, FD2D/dist), for $k \in \{-, 0, 10, 15\}$.

Figure 6.1: Representation and evaluation of the resulting Dist from simulations $s^1_k$, with $k \in \{-, 0, 10, 15\}$, considering the Bitcoin codebase.
Figure 6.2: Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 10, 15\}$, considering the Bitcoin codebase.
\( \gamma_e \). We speculate that \( sDists_2.selCAuthor \) is the reason behind this failure. By randomly picking a developer, all the developers make at least one commit in the early stages of development, which may not correspond to reality. To confirm this, through the analysis of \( newDevelopersUntilT \) aspect shown in Figure 6.3, we observe that our simulations reach the maximum number of developers in the early stages of development, while empirically there is a linear growth over time. Therefore, we advance to a new iteration similar to \( sDists_2 \), changing \( selCAuthor \).

![Figure 6.3: newDevelopersUntilTimeT aspect comparison between timeline generated for \( s^2 \) and empirical timeline, concerning Bitcoin codebase.](image)

### 6.3.3 Third Iteration

To construct \( selCAuthor \) for this iteration, i.e. choosing which developer should commit in a time step, it is necessary to consider if it will be a: (1) new developer; (2) developer that already committed. To decide between (1) and (2), we previously concluded that new developers appear uniformly over time. After this decision, a distribution to choose a specific developer needs to be constructed.

Various methods can be proposed to choose a developer for (1) and for (2). We consider the possibilities of either choosing randomly (\( RAND \)), or, depending on the time step \( t \), choosing a developer \( d \) with probability proportional to the number of commits \( d \) still needs to do from \( t \) onwards (\( PROP.TODO \)) or proportional to the number of commits \( d \) has done until \( t \) (\( PROP_DONE \)).

A description of the construction of this distribution in time step \( t \) can be seen in Listing 6.4, defining:

- \textit{generalRules} as a class of different methods to create a distribution based on a sequence of values.
- \textit{newDevR} and \textit{knownDevR} as rules to choose a developer for (1) and (2), respectively, which either can be \( RAND, PROP.TODO \) or \( PROP_DONE \).
\textbf{comPerDeveloper}_t0, \textbf{comPerDeveloper} as sequences representing the commits yet to do per developer in the first time step and in the actual time step, respectively. The index $i$ of \textbf{comPerDeveloper}_t0 sequence corresponds to the number of commits that developer $i$ needs to do over the entire simulation.

$\text{diff}(s1, s2)$ returns a subset of sequence $s1$ without the values present in sequence $s2$.

\textbf{Listing 6.4:} Commit author selection (selCAuthor) distribution construction pseudocode for third iteration.

```python
1 class generalRules {  # methods to create distribution
2     def random(sequence) {  
3         return [1 / len(sequence)] * len(sequence)
4     }
5
6     def probProportionalToValue(sequence) {  
7         dist = []
8         for i in range 1..len(sequence) {  
9             dist.add(sequence[i] / sum(sequence))
10             return dist
11         }
12     }
13 }
14
15 inputs:
16     string: newDevR, knownDevR
17     CodebaseAttributes: ca
18     sequence[developer]: comPerDeveloper_t0, comPerDeveloper
19 outputs: developer
20
21 allDevsCommitted = True
22 devsWithoutCommitsDone = []
23 for i in range 1..len(commitsPerDeveloper) {  
24     if comPerDeveloper[i] == comPerDeveloper_t0[i] {  
25         devsWithoutCommitsDone.add(i)
26         allDevsCommitted = False
27     }
28 }
29
30 if not allDevsHaveCommitted and random() < ca.nod / ca.noc: {  
31     # add new developers uniformly
32     devsToConsider = devsWithoutCommitsDone
33     methodToCreateDist = newDevR
34 }
35 else {  
36     allDevs = range 1..len(commitsPerDeveloper)
37     devsToConsider = diff(allDevs, devsWithoutCommitsDone)
38     methodToCreateDist = knownDevR
39 }
40
41 if methodToCreateDist == RAND {  
42     devsToConsiderDist = generalRules.random(devsToConsider)
43 }
44 else {  
45     devsToConsiderCommits = []
46     if methodToCreateDist == PROP_TODO {  
47         for i in range 1..len(devsToConsider): {  
48             developerI = devsToConsider[i]
49             devsToConsiderCommits.add(comsPerDeveloper[developerI])
50         }
51     }
52     else if methodToCreateDist == PROP_DONE {  
53         for i in range 1..len(devsToConsider) {  
54             developerI = devsToConsider[i]
55             devsToConsiderCommits[developerI].add(comsPerDeveloper[developerI])
56         }
57     }
58 }
```
commitsDoneByDevI = comPerDeveloper[0][developerI] - comPerDeveloper[developerI]

if commitsDoneByDevI == comPerDeveloper[0][developerI]:
    # has done all commits, can not do more commits
devsToConsiderCommits.add(0)
else:
    devsToConsiderCommits.add(commitsDoneByDevI)

devsToConsiderDist = generalRules.probProportionalToValue(devsToConsiderCommits)

allDevsDist = []
for i in range(len(commitsPerDeveloper)):
    if len(devsToConsider) > 0 and devsToConsider[0] == i:
        allDevsDist.add(devsToConsiderDist.prob(i))
        devsToConsider.pop()
    else:
        allDevsDist.add(0)

return allDevsDist

Considering the rules newDevR and knownDevR we analyse the possible sets of input, excluding the possibilities where newDevR = PROP_DONE, as the resultant distribution is similar to the case when newDevR = RAND, given that all the developers who have not committed have done zero commits. Therefore, sDists_j with j ∈ [3–8] are constructed similarly to sDists_2, but changing sDists_j.selCAuthor to be constructed through the use of Listing 6.4, where:

- j = 3 uses (newDevR = RAND, knownDevR = RAND)
- j = 4 uses (newDevR = RAND, knownDevR = PROP_TODO)
- j = 5 uses (newDevR = RAND, knownDevR = PROP_DONE)
- j = 6 uses (newDevR = PROP_TODO, knownDevR = RAND)
- j = 7 uses (newDevR = PROP_TODO, knownDevR = PROP_TODO)
- j = 8 uses (newDevR = PROP_TODO, knownDevR = PROP_DONE)

Given this, simulations s_j^k, with j ∈ [3–8] and k ∈ {−, 0, 10, 15} are ran. To analyse its results, we investigate: (1) Dist = D2F/F/dist, through: (1.1) D2F γ diff, which is the difference between the γ of the resulting D2F/F/dist and the empirical D2F/F/dist. In the case that the resulting D2F/F/dist does not have a heavy tail, D2F γ diff = −; (1.2) D2F KS, which is the KS between the resulting D2F/F/dist and the empirical D2F/F/dist; (2) Dist = FD2D/dist, through: (2.1) FD2D γ diff, similar to D2F γ diff, but for FD2D/dist; (2.2) FD2D KS, similar to D2F KS, but for FD2D/dist; (2.3) FD2D max diff, which is the
division between the maximum individual collaboration of FD2D/dist resulting from $s_{jk}^i$ by the maximum individual collaboration of the empirical FD2D/dist. In this case, 100% is the best result possible; (3) the part of CBAspEval that changes given different $j$: (3.1) $\text{newDevelopersUntil} T$ KS; (3.2) $\tilde{U}_{\text{global} \text{dist}}$ KS; $\tilde{U}_{\text{local} \text{dist}}$ KS. This whole analysis is defined as Complex Evaluation (CEval) from now on.

In Table 6.1 we show the CEval for $s_{jk}^i$, with $j \in [3-8]$ and $k \in \{-, 0, 10, 15\}$. Considering the parts of CBAspEval, we ignore results with KS $> 0.15$, making $s_{Dists}^5$ and $s_{Dists}^8$ the best choices. From analysis of results for D2F $\gamma$ diff, D2F KS, FD2D $\gamma$ diff and FD2D KS we can conclude that $s_{jk}^i$, with $j \in \{5, 8\}$ and $k \in \{10, 15\}$ reach near the empirical results, therefore we compare FD2D max diff. It is observable that $s_{jk}^i$ with $k \in \{10, 15\}$ has high difference in terms of FD2D max diff. Given this, $s_{Dists}^8$ is the preferable option.

<table>
<thead>
<tr>
<th>$j$</th>
<th>$k$</th>
<th>D2F $\gamma$ diff and KS</th>
<th>D2D $\gamma$ diff and KS</th>
<th>FD2D $\gamma$ diff KS</th>
<th>FD2D max diff</th>
<th>newDeveloper until T KS</th>
<th>$\tilde{U}_{\text{global} \text{dist}}$ KS</th>
<th>$\tilde{U}_{\text{local} \text{dist}}$ KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-</td>
<td>(-, 0.36)</td>
<td>1.4e-01</td>
<td>0.15</td>
<td>37</td>
<td>0.12</td>
<td>0.09</td>
<td>0.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.58, 0.29)</td>
<td>3.2e-01</td>
<td>0.59</td>
<td>139</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.05, 0.17)</td>
<td>1.7e-01</td>
<td>0.22</td>
<td>46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.28, 0.22)</td>
<td>1.6e-01</td>
<td>0.2</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.60, 0.26)</td>
<td>8.0e-02</td>
<td>0.66</td>
<td>230</td>
<td>0.13</td>
<td>0.04</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>(0.05, 0.16)</td>
<td>1.0e-01</td>
<td>0.25</td>
<td>95</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>(0.13, 0.21)</td>
<td>1.1e-01</td>
<td>0.21</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>(-, 0.36)</td>
<td>1.9e-01</td>
<td>0.15</td>
<td>19</td>
<td>0.07</td>
<td>0.09</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.59, 0.26)</td>
<td>2.2e-01</td>
<td>0.64</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>(0.01, 0.18)</td>
<td>3.0e-01</td>
<td>0.25</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>(0.23, 0.23)</td>
<td>2.7e-01</td>
<td>0.19</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>-</td>
<td>(-, 0.35)</td>
<td>1.2e-01</td>
<td>0.15</td>
<td>44</td>
<td>0.16</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.64, 0.29)</td>
<td>4.0e-02</td>
<td>0.59</td>
<td>381</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.04, 0.17)</td>
<td>5.0e-02</td>
<td>0.21</td>
<td>94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.36, 0.20)</td>
<td>8.0e-02</td>
<td>0.16</td>
<td>71</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>-</td>
<td>(-, 0.35)</td>
<td>3.0e-02</td>
<td>0.15</td>
<td>110</td>
<td>0.15</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.61, 0.28)</td>
<td>7.0e-02</td>
<td>0.78</td>
<td>517</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.03, 0.17)</td>
<td>1.0e-02</td>
<td>0.22</td>
<td>202</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.12, 0.21)</td>
<td>1.0e-02</td>
<td>0.18</td>
<td>165</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>(-, 0.35)</td>
<td>2.0e-02</td>
<td>0.15</td>
<td>71</td>
<td>0.13</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.58, 0.27)</td>
<td>6.0e-02</td>
<td>0.74</td>
<td>352</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.12, 0.18)</td>
<td>3.0e-02</td>
<td>0.2</td>
<td>141</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.31, 0.22)</td>
<td>1.0e-02</td>
<td>0.16</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Figure 6.4 we represent, per different Dist, the results of $s_{jk}^i$ in comparison with the empirical results alongside the respective Evals.

For Dist $\in \{D2F/F/dist, F2F/dist\}$ the results are similar to the ones described in the first and second iterations, where $s_{10}^8$ leads to the Dist that have more similarities with the empirical Dist.

As in the second iteration, for Dist = FD2D/dist, all the simulations achieve the non-trivial property...
Figure 6.4: Representation and evaluation of the resulting Dist from simulations $s^k_n$, with $k \in \{-, 0, 10, 15\}$, considering the Bitcoin codebase.
of the empirical distribution. Even though some simulations’ distribution is closer in terms of KS to the empirical distributions, all simulations have $\gamma_t$ similar to $\gamma_e$, namely the simulation using the null hypothesis, which means that the chance method leads to non-trivial properties.

We conjecture that the reason behind this property is related to the nature of the system. As some developers stand out in terms of the number of commits they do, these developers have more possibilities of collaboration, making them also stand out in terms of collaboration, even if they choose files randomly, i.e. through the null hypothesis.

Given that files are chosen randomly in $s_{k}^n$, collaboration is done uniformly across files, with some containing more collaboration as they existed for more time. Considering Collaboration Done Per File Distribution (CollabPerFile/dist), we can observe in Figure 6.5 that CollabPerFile/dist resulting from $s_{k}^n$ is completely dissimilar to the empirical CollabPerFile/dist, as the former is described by an exponential distribution and the latter is described by a power law distribution. Taking this analysis into account and that $s_{k}^n$ does not reach the non-trivial properties for $\text{Dist} \in \{ \text{D2F/F/dist, F2F/dist} \}$, we conclude that the null hypothesis is not sufficient to explain the collaboration process in the codebase.

![Figure 6.5: Representation of CollabPerFile/dist resulting from simulations $s_{k}^n$, with $k \in \{-, 0, 10, 15\}$ in comparison with empirical CollabPerFile/dist, for Bitcoin codebase.](image)

Concerning the distribution of collaboration per file, we observe that the distribution resulting from $s_{10}^n$ is similar to the empirical distribution. Given this, and considering also that $\text{Dist} \in \{ \text{D2F/F/dist, F2F/dist, FD2D/dist} \}$ resulting from $s_{10}^n$ have the most similarities with empirical Dist, hypothesis $h_{10}$ together with $s_{\text{Dist}10}$ is a reliable set of inputs for Listing 6.3 to achieve results similar to the ones shown for the empirical codebase. To confirm this, we investigate the CBAspEval between the theoretical timeline resultant from the algorithm described in Listing 6.1 and the empirical timeline. The aspects comparison related to commits and developers can be seen in Figure 6.6. The other aspects, related
with files interactions and dates, are represented in Figure 6.7. For each comparison, KS can be seen in Table 6.2. Through the visual and statistical analysis, we conclude that the theoretical timeline generated with support from $sDist_{8S}$ has several similarities with the empirical timeline, as $KS \leq 0.21$, for all the aspects.

A similar analysis is done for the other codebases in Appendix A. We conclude that neither the best $sDist_{8j}$ nor $h_k$ are universal across the distinct codebases. The results of the best $sDist_{8j}$ and $h_k$ per codebase can be seen in Table 6.3. In this Table, we also add the mean changes per file as a conjecture to explain $k$ per codebase. Considering that $k$ represents the initial attractiveness of each file, $k$ will be higher when the minimum of the set of the importance of all files is higher. This conjecture is justified by the fact that in our model, the importance of a file is given by the number of changes it receives, so we hypothesize that $k$ is higher when the average number of changes per file is higher. Although this is true, Kubernetes and Tensorflow have similar mean changes per file and they are simulated through different
Figure 6.7: Comparison between codebase aspects related with files and dates of CBAspEval between the empirical timeline of the Bitcoin codebase and the theoretical timeline generated with support from $sDists_8$ for Bitcoin codebase.

Table 6.2: CBAspEval between the empirical timeline of the Bitcoin codebase and the theoretical timeline generated in Listing 6.1 with support from $sDists_8$ for Bitcoin codebase.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitsPerDevDist</td>
<td>0.05</td>
</tr>
<tr>
<td>newDevelopersUntilT</td>
<td>0.12</td>
</tr>
<tr>
<td>$U_{global}dist$</td>
<td>0.09</td>
</tr>
<tr>
<td>$U_{local}dist$</td>
<td>0.10</td>
</tr>
<tr>
<td>filesAddedUntilT</td>
<td>0.20</td>
</tr>
<tr>
<td>filesChangedUntilT</td>
<td>0.04</td>
</tr>
<tr>
<td>filesDeletedUntilT</td>
<td>0.08</td>
</tr>
<tr>
<td>timePassedUntilT</td>
<td>0.21</td>
</tr>
</tbody>
</table>
Table 6.3: Comparison between the best inputs for each codebase and the empirical average changes per file, for different codebases.

<table>
<thead>
<tr>
<th>Codebase</th>
<th>Best $sDists_j$</th>
<th>Best $h_k$</th>
<th>Mean changes per file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ansible</td>
<td>$sDists_8$</td>
<td>$h_{2.5}$</td>
<td>2.74</td>
</tr>
<tr>
<td>Bitcoin</td>
<td>$sDists_8$</td>
<td>$h_{10}$</td>
<td>5.75</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>$sDists_8$</td>
<td>$h_5$</td>
<td>4.38</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>$sDists_5$</td>
<td>$h_{10}$</td>
<td>4.25</td>
</tr>
<tr>
<td>Terraform</td>
<td>$sDists_8$</td>
<td>$h_{2.5}$</td>
<td>2.23</td>
</tr>
</tbody>
</table>

Given this, and answering RQ2, we conclude that the emergent patterns of collaboration and other patterns related to the actors that intervene in the collaboration process can be simulated by a rather simple mechanism where developers choose files to change according to the rule described in Equation (2.6) with $A$ being similar in all files. This target is achieved across various codebases, with $A$ being different across them, and showing some correlation with the empirical average changes per file of each inherent real codebase. In addition to this mechanism, the construction of the timeline differs across the distinct codebases: (1) considering Ansible, Bitcoin, Kubernetes and Terraform, a set of distributions previously defined as $sDists_8$ is used, where: (1.1) the distribution of the number of commits per developer follows a power law distribution; (1.2) new developers are added uniformly over time, considering a specific rule to choose which developer to commit in a certain time step $t$, described by: (1.2.1) the probability of choosing a new developer $d$ is proportional to the number of commits $d$ has to do; (1.2.2) when choosing developers who have already committed until $t$, the probability of choosing a developer $d$ is proportional to the number of commits that $d$ has done until $t$; (2) in respect with Tensorflow, the construction of the timeline is similar to (1), aside from changing (1.2.1) to choose randomly a developer that has not yet committed.
7 Discussion

Contents

7.1 Software Engineering Considerations ............................................. 52
7.2 Threats to Validity ................................................................. 55
In this Chapter, we discuss the meaning of our results in Software Engineering, namely on the structure of teams and modules. After this, we describe the threats to the validity of our work.

7.1 Software Engineering Considerations

By using complex networks, we investigate the patterns and their origins concerning a real system, more specifically, the social system of software. There are key concepts of this system that should be discussed considering our results. The first are files and the aggregates of files commonly named as modules. Normally, files are either grouped in modules to facilitate the development of certain functionality or to limit the propagation of changes. The second are developers and the aggregates of developers which are commonly named as teams.

Regarding D2F/F/dist, a small number of files are highlighted against the rest of the files in terms of changes. We made an ad-hoc analysis of the top files of each repository to understand their detachment from the rest of the files. The results can be seen in Table 7.1. For each codebase, we show the fourth most changed files, with their number of changes and functionality. For simplicity, we created subgroups of functionality to facilitate the analysis: (1) Documentation; (2) Business, which is related to the business logic; (3) Utils, which is non-business logic which is shared amongst various files, such as hash algorithms, file system management, etc.; (3) Dependencies, which are files responsible for managing the packages dependencies in the repository; (4) Build, responsible for module building, i.e. transforming source code files into software artifacts; (5) Tests, responsible for testing the codebase. Through a first analysis, we can see that the top files are codebase-dependent. More specifically, Bitcoin is highlighted due to all of its top files being related to the business; Tensorflow is highlighted as it is the only codebase with building files being the most changed; Terraform is the only codebase that has a file with higher changes relative to tests. Independently of the codebase, we can conclude that all of the files are core files as they keep the key concepts of each codebase, either by documenting it, being key business logic, managing dependencies or building the modules.

Considering F2F/dist, a file $f$, and the sum of files across commits where $f$ is changed as $relationsNumber(f)$, a small number of files are highlighted in relation to others in terms of the function $relationsNumber$. This implies that these files are highly coupled with others. The description of these files is shown in Table 7.2, similarly as in Table 7.1. As before, we can conclude that the most coupled files are codebase-dependent. From the Table, it is possible to conclude that the main functionalities of the most coupled files are Business and Dependencies. Concerning Business, we can conclude that those files are key business logic of the codebase. Concerning Dependencies, they are the most coupled as they are the strings keeping the codebase together. In particular, for Tensorflow, the most coupled files are related to the build of the code, which is inherent to the business logic of the codebase.
Table 7.1: Top 4 most changed files for the various analysed codebases and their functionalities.

<table>
<thead>
<tr>
<th>Codebase</th>
<th>Files</th>
<th>Changes</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>ansible</td>
<td>CHANGELOG.md</td>
<td>1055</td>
<td>Documentation</td>
</tr>
<tr>
<td></td>
<td>runner/<em>init</em>.py</td>
<td>667</td>
<td>Business (API)</td>
</tr>
<tr>
<td></td>
<td>.github/BOTMETA.yml</td>
<td>636</td>
<td>Documentation</td>
</tr>
<tr>
<td></td>
<td>module_utils/basic.py</td>
<td>459</td>
<td>Utility</td>
</tr>
<tr>
<td>bitcoin</td>
<td>src/validation.cpp</td>
<td>1907</td>
<td>Business</td>
</tr>
<tr>
<td></td>
<td>src/init.cpp</td>
<td>1344</td>
<td>Business</td>
</tr>
<tr>
<td></td>
<td>src/wallet/wallet.cpp</td>
<td>1306</td>
<td>Business</td>
</tr>
<tr>
<td></td>
<td>src/net.cpp</td>
<td>982</td>
<td>Business</td>
</tr>
<tr>
<td>kubernetes</td>
<td>kubelet/kubelet.go</td>
<td>1452</td>
<td>Business</td>
</tr>
<tr>
<td></td>
<td>framework/util.go</td>
<td>1001</td>
<td>Utility</td>
</tr>
<tr>
<td></td>
<td>core/types.go</td>
<td>840</td>
<td>Business (API)</td>
</tr>
<tr>
<td></td>
<td>validation/validation.go</td>
<td>786</td>
<td>Dependencies</td>
</tr>
<tr>
<td>tensorflow</td>
<td>python/BUILD</td>
<td>1319</td>
<td>Build (Python)</td>
</tr>
<tr>
<td></td>
<td>tensorflow/workspace.bzl</td>
<td>1238</td>
<td>Dependencies</td>
</tr>
<tr>
<td></td>
<td>core/BUILD</td>
<td>1124</td>
<td>Build (Core)</td>
</tr>
<tr>
<td></td>
<td>kernels/BUILD</td>
<td>1007</td>
<td>Build (Kernels)</td>
</tr>
<tr>
<td>terraform</td>
<td>CHANGELOG.md</td>
<td>3086</td>
<td>Documentation</td>
</tr>
<tr>
<td></td>
<td>CHANGELOG.md</td>
<td>775</td>
<td>Documentation</td>
</tr>
<tr>
<td></td>
<td>terraform/context_apply_test.go</td>
<td>409</td>
<td>Tests (Context)</td>
</tr>
<tr>
<td></td>
<td>vendor/vendor.json</td>
<td>380</td>
<td>Dependencies</td>
</tr>
</tbody>
</table>

Ansible is the only one where one of the most coupled files is related to Tests. In this specific case, it serves as a sanity to verify if the codebase functionality is working correctly, making it highly coupled as it tests the different functionalities of the codebase. Regarding the files for the different codebases, we can conclude that they are core files of the codebase, being characterized either by being key business logic or by the ability to keep various functionalities of the codebase together.

Regarding developers, From the results of D2F/D/dist, we found that a small number of developers are highlighted in relation to the other developers in terms of the number of changes that they made. Through an ad-hoc investigation, these members are generally characterized as the lead maintainers/architects of such codebases, being essential for the decisions taken, and so, characterized as core developers. Similar results were found for FD2D/dist, observing that there is a small number of developers which are highlighted with respect to the rest of the developers as they collaborate much more than others, with them being characterized as the lead maintainers/architects of such codebases. Other analysis should be conducted to verify if this group of developers are a strong core team in the codebase and how related they are with the other developers.

The results of the simulation coincide with some ideas discussed before. It was found that it is sufficient for developers to choose files based on a rule of preferential attachment concerning the changes of the file plus some initial attractiveness of each file. From this, it is concluded that the concepts of modules and teams are not explicitly needed to simulate the observed empirical patterns and that given the initial attractiveness, each file can be potentially changed at each point in time, therefore every file
Table 7.2: Top 4 most coupled files for the various analysed codebases and their functionalities.

<table>
<thead>
<tr>
<th>Codebase</th>
<th>Files</th>
<th>Relations</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>ansible</td>
<td>sanity/ignore.txt</td>
<td>5585</td>
<td>Tests</td>
</tr>
<tr>
<td></td>
<td>action/_init_py</td>
<td>2537</td>
<td>Business</td>
</tr>
<tr>
<td></td>
<td>executor/task_executor.py</td>
<td>2474</td>
<td>Business</td>
</tr>
<tr>
<td></td>
<td>strategy/_init_py</td>
<td>2029</td>
<td>Business</td>
</tr>
<tr>
<td>bitcoin</td>
<td>src/validation.cpp</td>
<td>7596</td>
<td>Business</td>
</tr>
<tr>
<td></td>
<td>src/init.cpp</td>
<td>7240</td>
<td>Business</td>
</tr>
<tr>
<td></td>
<td>src/wallet_wallet.cpp</td>
<td>6845</td>
<td>Business</td>
</tr>
<tr>
<td></td>
<td>src/wallet_wallet.h</td>
<td>4764</td>
<td>Business</td>
</tr>
<tr>
<td>kubernetes</td>
<td>vendor/modules.txt</td>
<td>18515</td>
<td>Dependencies</td>
</tr>
<tr>
<td></td>
<td>go.mod</td>
<td>14989</td>
<td>Dependencies</td>
</tr>
<tr>
<td></td>
<td>openapi-spec/swagger.json</td>
<td>14867</td>
<td>Business (API)</td>
</tr>
<tr>
<td></td>
<td>Godeps/Godeps.json</td>
<td>14266</td>
<td>Dependencies</td>
</tr>
<tr>
<td>tensorflow</td>
<td>python/BUILD</td>
<td>11741</td>
<td>Build</td>
</tr>
<tr>
<td></td>
<td>core/BUILD</td>
<td>9855</td>
<td>Build</td>
</tr>
<tr>
<td></td>
<td>kernels/BUILD</td>
<td>8502</td>
<td>Build</td>
</tr>
<tr>
<td></td>
<td>gpu/BUILD</td>
<td>6888</td>
<td>Build</td>
</tr>
<tr>
<td>terraform</td>
<td>vendor/vendor.json</td>
<td>5246</td>
<td>Dependencies</td>
</tr>
<tr>
<td></td>
<td>go.sum</td>
<td>2983</td>
<td>Dependencies</td>
</tr>
<tr>
<td></td>
<td>go.mod</td>
<td>2900</td>
<td>Dependencies</td>
</tr>
<tr>
<td></td>
<td>vendor/modules.txt</td>
<td>2388</td>
<td>Dependencies</td>
</tr>
</tbody>
</table>

has importance in the codebase, which corresponds to reality. Considering the former conclusion, this does not imply that using modules explicitly does not lead to the observed empirical patterns. An analogous algorithm to the one defined in Listing 6.3 could be constructed, where modules would be defined explicitly. This simulation is not trivial given that, by having different objectives, modules may have different sizes and behaviours through the codebase evolution, i.e. the size and relations are not constant through time, which would involve a thorough analysis of such aspects. Our algorithm primes for its simplicity and its success in simulating the observed patterns without defining such restrictive collective ideas. The patterns originate simply from the local interactions of developers, without a central authority restraining the decisions of developers.

Even though the concept of modules is not explicitly needed to simulate such patterns, it does not mean that the collection of software artifacts lacks a structure. Due to the preferential attachment rule, there exists a certain group of files which have more changes in relation to the others. Further analysis of the results of the simulation should be conducted to determine if this group can be established as a core module, given that this was also discussed when describing the empirical results. Besides this, considering the files with fewer changes, this group should be investigated, to determine if these files form a periphery which is highly connected with a possible core, lacking internal connections, or if they form a substantial network of connections between themselves. The same reasoning goes for developers and their collaborations, to investigate the existence of a core team, and if the developers with less collaboration form a peripheral group lacking internal connections and being well connected.
with the core team. In summary, the existence of a core-periphery structure should be analysed in F2F and FD2D. It should be considered that all of the analysed projects are open-source and different results may appear in closed-source software.

### 7.2 Threats to Validity

The threats to the validity of this work are related to the modelling of the system and the data gathering complexity.

#### 7.2.1 System Modelling

Considering the modelling of the system, two factors are threats to validity: (1) the interpretation of collaboration; (2) the interpretation of coupling.

Regarding (1), as discussed in the Related Work section, although their work is in the same context, developers may not collaborate when changing the same software module, as this type of collaboration is not direct since developers who change the same source code file do not exchange direct messages. Therefore, the validity of studies is questioned. Diverse works have studied this validity across different networks, with some concluding that these networks are supported by developers’ perceptions while others conclude otherwise. This will remain a threat to the validity of our work, as this is out of the scope of our work.

However, as it can be argued that using files instead of methods may lead to extra erroneous interpretations of collaboration between developers, given the formers’ larger context, we will compare both approaches.

For this purpose, we construct a network similar to FD2D, however, two developers are connected if there is a pair of commits, one from each developer, spaced at maximum by a month, to the same method. The connection is measured with an identical weighting process to FD2D except for considering methods instead of files in Equation (4.1). This network will be defined as MD2D and is constructed based on D2M.

More particularly, we investigate the existence of a correlation per developer, of its degree in FD2D and MD2D. If a correlation is found, we can argue that describing the emergent patterns of collaboration by considering files interactions is a preferred method in relation with describing the emergent patterns of collaboration by taking into account the interactions in methods, as both are correlated and the latter is more complicated to compute, programming-language dependent and not suitable for heterogenous documents, such as source-code and configuration files [40].

In Figure 7.1 it is possible to observe the relation between the WD of a developer in FD2D and MD2D, across the analysed codebases.
Figure 7.1: Comparison, per developer, of its WD in FD2D and MD2D across the different analysed codebases.
By visual inspection, we can observe some correlation between the empirical data. Statistically speaking, by fitting a power law, the results per codebase can be seen in its respective figure, or in Table 7.3. From these results, we can observe a strong correlation ($\sigma < 0.05$), approximately uniform ($\gamma \sim 1$) for all codebases. Given this similarity, we can conclude that using FD2D is advantageous over using MD2D to analyse emergent patterns.

Table 7.3: Power law fitting to the relation between the WDs per developer in FD2D and MD2D across the different analysed codebases.

<table>
<thead>
<tr>
<th>Codebase</th>
<th>$\gamma$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ansible</td>
<td>1.164</td>
<td>0.013</td>
</tr>
<tr>
<td>Bitcoin</td>
<td>1.15</td>
<td>0.017</td>
</tr>
<tr>
<td>Kubernetes</td>
<td>1.137</td>
<td>0.011</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>0.917</td>
<td>0.012</td>
</tr>
<tr>
<td>Terraform</td>
<td>1.035</td>
<td>0.024</td>
</tr>
</tbody>
</table>

Concerning the coupling of the files, we assume that files in the same commit are coupled with each other. This may not be true as there is not a clear restriction that files in the same commit should be related, however, given that commits need to be peer-reviewed to be accepted into the codebase, the group of the files in a commit are chosen carefully, which is generally justified by their relations, i.e. coupling. In summary, generally, if files are in the same commit, it implies that a change in one or more of them was propagated to the remaining files.

### 7.2.2 Data Gathering

Considering the gathering and pre-processing of data, various threats exist, namely considering developers’ identity, the impact of merge commits and the methods renames.

Some authors suggest that a developer can use different identities to make changes in the codebase [30, 41], proposing procedures to minimize these errors. In our analysis, we consider each different GitHub account as a different identity given that these proposed procedures in [30, 41] may also lead to mismatches and require manual inspection, which is also fallible. This is a threat to validity as we may be considering one developer as two or more different developers, in the case that a developer uses more than one GitHub account. In practical terms, as the identification of developers in a codebase is crucial, for example, to identify who is responsible for a certain area, it would only add more complexity if different developers needed to be labelled to the same person. Given this, we assume that this event is rare and that considering a GitHub account as a unique developer has a small impact on the identification of developers concerning reality.

As discussed in Section 2.1, the existence of merge commits in VCS needs to be considered to conduct a proper analysis of Software Development. In our work, we do not consider merge commits, as normally, they do not make any original changes. However, this remains a threat to validity as there
may be some merge commits that are exceptional to this rule. We conjecture that such exceptions are rare and do not have an impact on our work.

Finally, considering method renames, in our work we do not relabel any methods if their name is changed given the problem of non-determinism. Considering a developer $d_1$ and a developer $d_2$ and a method $m$, if $d_1$ changes $m$ renaming it, when $d_2$ changes $m$, it will be not possible to consider the collaboration between $d_1$ and $d_2$. Therefore, some information on methods collaboration is lost when a method is renamed leading to a threat to validity. Further analysis should be done to understand the implication of this choice.
8

Conclusion

Contents

8.1 Conclusions .......................................................... 60
8.2 Future Work .......................................................... 60
8.3 Thesis Artifacts ....................................................... 61
In this Chapter we summarize both research questions and their answers, discussing the contributions of this work. After this, we discuss possible future work.

8.1 Conclusions

To simulate the emergent patterns of developers' collaboration beneath the interactions between developers and files, as questioned by the second research question, it is necessary to describe the actors' emergent patterns, as formulated by the first research question. Through the construction and analysis of three networks aiming to represent the actual interactions between such actors, we conclude that, generally, these patterns are described by power law distributions, which means that a small number of actors are highlighted in relation to their pairs.

To answer the second research question, we develop a general algorithm that reproduces the interactions between developers and files. By iterative development of the algorithms' input with support of the results of the first research question, the emergent patterns are reached through a simple set of rules. Regarding the timeline, it is sufficient that: (1) the number of commits per developer follows a power law distribution; (2) new developers are added uniformly over time; (3) when developers are added, either a developer is added randomly or the probability of choosing a new developer $d$ is proportional to the number of commits $d$ has to do, depending on the codebase; (4) in a determined time step $t$, when selecting a developer $d$ that already committed, the probability of choosing $d$ is proportional to the number of commits $d$ has done until $t$. Considering how developers choose files, we find that it is sufficient that developers change a file $i$ with probability proportional to an initial attractiveness of $i$ plus the number of changes $i$ already received. The value of the initial attractiveness is codebase-dependent, having some correlation with the average changes done per file in each respective codebase.

This is the first time that such emergent patterns are simulated. Two key discoveries are highlighted: (1) different open-source codebases patterns can be simulated by rather similar rules; (2) to simulate such patterns, it is not needed to consider other key concepts of software development such as teams, i.e. aggregates of developers, or modules, i.e. aggregates of files.

8.2 Future Work

For future work, diverse paths are possible. We leave open problems in terms of the study of the evolution of the emergent patterns for the actors involved in the collaboration process and concerning some aspects of timeline generation, amongst others. Besides this, new paths can be taken. Regarding the networks discussed in Chapter 3 to model collaboration between developers, a comparison between the different networks should be done to understand their differences and similarities. Considering our
simulation model, more real codebases can be analysed to investigate if the observed mechanisms are similar to those that we found. Our work focus on developers’ collaboration, however, further investigation should be done on F2F network and which mechanisms lead to its emergent patterns. It is also possible to study hypothetical codebases to understand which codebase attributes lead to emergent patterns similar to those we found. Finally, the investigation of a correlation between the success of an open-source codebase and the rules that are sufficient for its simulation should be conducted, to verify if prosperous codebases derive from certain mechanisms.

8.3 Thesis Artifacts

As contributions of this thesis, we provide the used code and its results in the Ubuntu 18.04.4 LTS environment.
Bibliography


In this Appendix, we describe the simulations for Ansible, Kubernetes, Tensorflow and Terraform codebases. We start by discussing the similarities and differences between the simulations for these codebases. Considering the similarities of the iterations: (1) each of these codebases follow the first two iterations in a similar manner as discussed for Bitcoin, exactly for the same reasons: (1.1) the first iteration, namely characterized by $s_{Dists_1}$, leads to simulations $s^1_k$ that do not achieve the non-trivial properties shown by the empirical $FD2D/dist$, failing due to the use of uniform $nocPerDevDist$ for Object InputDistributions described in Listing 6.1, while in reality, the number of commits per developer follows a power law distribution in the distinct codebases; (1.2) the second iteration, characterized by $s_{Dists_2}$, lead to simulations $s^2_k$ whose $Dist = FD2D/dist$ reach non-trivial properties similar to the empirical $Dist$, however this $Dist$ has $\gamma_t \sim \gamma_e$ and high KS in relation with the respective empirical $FD2D/dist$. The reason for failure is similar to in Bitcoin, as a uniform distribution is used for $selCAuthor$ for Object InputDistributions described in Listing 6.1, which does not correspond to the real choice of authors; (2) as in Bitcoin, the results for $Dist \in \{D2F/F/dist, F2F/dist\}$ are similar across all iterations, with all hypothesis except the null one reaching the non-trivial properties displayed by the respective empirical $Dist$. Some hypotheses are preferable given that they reach $Dist$ with $\gamma_t \sim \gamma_e$ and small KS, with these hypotheses
being dependent on each codebase. For the different codebases, note also that, as in Bitcoin, for Dist = F2F/dist, KS is high due to the similarity of the empirical F2F/dist with a Lomax distribution, which is not achieved by the distinct $s_j$.

The differences for these codebases simulation are: (1) their hypotheses; (2) the $sDist s_j$ used for the third iteration, where $j \in [3 – 8]$; (3) their results for the codebase aspects comparison. Their hypothesis differs according to the following:

- Ansible and Terraform use $h_-, h_0, h_{2.5}, h_5$
- Kubernetes uses $h_-, h_0, h_5, h_{10}$
- Tensorflow uses $h_-, h_0, h_{10}, h_{15}$

It should be noted for the difference (2) that while $j$ is different across codebases, and that $s_j$ leads to a Dist = FD2D/dist characterized by its non-trivial properties, similarly to Bitcoin, $s_j$ is never sufficient to explain the collaboration process that is happening in the empirical codebase as Dist $\in \{D2F/F/dist, F2F/dist\}$ and CollabPerFile/dist resulting from $s_j$ are always distinct from the empirical Dist and CollabPerFile/dist, respectively.

Until the end of this Annex, we represent the results of each codebase grouped by iteration, highlighting specific similarities and differences. For each different analysis, a section is created to support readability. From now on, consider $RCB \in \{Ansible, Kubernetes, Tensorflow, Terraform\}$ and $h_{rcb}$ as the hypotheses for each RCB.
A.1 First Iteration

For the first iteration, the results for $s_k^1$, with $k \in h_{rcb}$ can be observed, per Dist, in Figures A.1 to A.4, respectively, for each RCB.

Figure A.1: Representation and evaluation of the resulting Dist from simulations $s_k^1$, with $k \in \{-0, 2.5, 5\}$, considering the Ansible codebase.
Figure A.2: Representation and evaluation of the resulting Dist from simulations \( s^*_k \), with \( k \in \{-0, 5, 10\} \), considering the Kubernetes codebase.
Figure A.3: Representation and evaluation of the resulting Dist from simulations $s^k$, with $k \in \{-0, 10, 15\}$, considering the Tensorflow codebase.
Figure A.4: Representation and evaluation of the resulting Dist from simulations $s^1_k$, with $k \in \{-0, 2.5, 5\}$, considering the Terraform codebase.
A.2 Second Iteration

Considering the second iteration, the representation, per Dist, of $s^2_k$ results, with $k \in h_{rcb}$ alongside the different Evals is shown in Figures A.5 to A.8, respectively, for each RCB.

![Diagram](image)

(a) Representation of D2F/dist resulting from simulations $s^2_k$, with $k \in \{-0.5, 0.0, 2.5, 5.0\}$ in comparison with empirical D2F/dist.

![Diagram](image)

(b) Evals ($s^2_k$, D2F/dist) with $k \in \{-0.5, 0.0, 2.5, 5.0\}$.

![Diagram](image)

(c) Representation of F2F/dist resulting from simulations $s^2_k$, with $k \in \{-0.5, 0.0, 2.5, 5.0\}$ in comparison with empirical F2F/dist.

![Diagram](image)

(d) Evals ($s^2_k$, F2F/dist) with $k \in \{-0.5, 0.0, 2.5, 5.0\}$.

![Diagram](image)

(e) Representation of FD2D/dist resulting from simulations $s^2_k$, with $k \in \{-0.5, 0.0, 2.5, 5.0\}$ in comparison with empirical FD2D/dist.

![Diagram](image)

(f) Evals ($s^2_k$, FD2D/dist) with $k \in \{-0.5, 0.0, 2.5, 5.0\}$.

Figure A.5: Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-0.5, 0.0, 2.5, 5.0\}$, considering the Ansible codebase.
Figure A.6: Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-, 0, 5, 10\}$, considering the Kubernetes codebase.
Figure A.7: Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-0, 0, 10, 15\}$, considering the Tensorflow codebase.
Figure A.8: Representation and evaluation of the resulting Dist from simulations $s^2_k$, with $k \in \{-0.25, 0.5\}$, considering the Terraform codebase.
A.3 Complex Evaluation

As in Bitcoin, we represent the CEval for $s^j_k$ with $j \in [3-8]$ and $k \in h_{rcb}$, for each RCB, in Tables A.1 to A.4. We ignore $s^j_k$ that have at least a part of CBApEval whose KS $> 0.15$. For Ansible, the best choices are $s_{Dist5}$ and $s_{Dist8}$. When analyzing the results of $s^j_k$, with $j \in \{5,8\}$ and $k \in \{2.5\}$, we can conclude that $D2F \gamma$ diff, $D2F$ KS, $FD2D \gamma$ diff and $FD2D$ KS reach near the empirical results. However, by observing $D2D$ max diff, we conclude that $s_{Dist8}$ is preferable to $s_{Dist5}$ as the latter maximum collaboration is only 16% of the empirical maximum collaboration. Considering Kubernetes, $s_{Dist8}$ is the best option given that for the different comparisons of CEval, it is similar to the empirical results, namely with $h_5$ and $h_{10}$. It should be noted that for any $s_{Dist_j}$, with $j \in [3-8]$, $FD2D \gamma$ diff is relatively high, which is related to the fact that the empirical Kubernetes $FD2D/dist$ is characterized by a double power law, while $FD2D/dist$ resulting from $s^j_k$ is not. With respect to Tensorflow, none $s_{Dist_j}$, with $j \in [3-8]$, has KS $< 0.15$ for the different analyzed codebase aspects. The best option is $s_{Dist8}$ which has the most similarities in terms of $D2F \gamma$ diff, $D2F$ KS, $FD2D \gamma$ diff, $FD2D$ KS, and namely $FD2D$ max diff. Finally, for Terraform, the preferable option is $s_{Dist8}$ given that it is the most similar with the empirical results across all different comparisons, mainly with $h_{2.5}$. In summary, for Ansible, Kubernetes and Terraform the best $s_{Dist_j}$ is $s_{Dist8}$ and for Tensorflow, $s_{Dist5}$ is the preferable option. We define $j_{rcb}$ as each RCB’s best option for $j$. 

76
Table A.1: CEval of $s_j^k$, with $j \in [3 - 8]$ and $k \in \{-, 0, 2.5, 5\}$, for Ansible codebase.

<table>
<thead>
<tr>
<th>$j$</th>
<th>$k$</th>
<th>D2F $\gamma$ diff and KS</th>
<th>FD2D $\gamma$ diff</th>
<th>FD2D KS</th>
<th>FD2D max diff</th>
<th>newDeveloper untilT KS</th>
<th>$U_{global, dist}$ KS</th>
<th>$U_{local, dist}$ KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-</td>
<td>(-, 0.32)</td>
<td>6.0e-02</td>
<td>0.22</td>
<td>26</td>
<td>0.1</td>
<td>0.05</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.30, 0.05)</td>
<td>3.0e-01</td>
<td>0.54</td>
<td>119</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.38, 0.17)</td>
<td>9.0e-02</td>
<td>0.22</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.91, 0.22)</td>
<td>8.0e-02</td>
<td>0.22</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>(-, 0.32)</td>
<td>4.0e-02</td>
<td>0.22</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.24, 0.06)</td>
<td>1.6e-01</td>
<td>0.56</td>
<td>100</td>
<td></td>
<td>0.07</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.30, 0.18)</td>
<td>1.6e-01</td>
<td>0.22</td>
<td>30</td>
<td></td>
<td>0.07</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.79, 0.22)</td>
<td>8.0e-02</td>
<td>0.22</td>
<td>24</td>
<td></td>
<td>0.07</td>
<td>0.08</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>(-, 0.32)</td>
<td>0.0e+00</td>
<td>0.22</td>
<td>12</td>
<td></td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.29, 0.05)</td>
<td>1.5e+00</td>
<td>0.55</td>
<td>35</td>
<td></td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.35, 0.18)</td>
<td>7.0e-02</td>
<td>0.22</td>
<td>16</td>
<td></td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.68, 0.23)</td>
<td>4.0e-02</td>
<td>0.22</td>
<td>14</td>
<td></td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>(-, 0.32)</td>
<td>2.3e-01</td>
<td>0.22</td>
<td>13</td>
<td></td>
<td>0.04</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.33, 0.05)</td>
<td>1.0e-01</td>
<td>0.53</td>
<td>160</td>
<td></td>
<td>0.04</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.31, 0.18)</td>
<td>2.2e-01</td>
<td>0.22</td>
<td>24</td>
<td></td>
<td>0.04</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.78, 0.23)</td>
<td>2.9e-01</td>
<td>0.22</td>
<td>15</td>
<td></td>
<td>0.04</td>
<td>0.08</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>(-, 0.32)</td>
<td>9.0e-02</td>
<td>0.22</td>
<td>34</td>
<td></td>
<td>0.04</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.33, 0.05)</td>
<td>1.0e-02</td>
<td>0.54</td>
<td>283</td>
<td></td>
<td>0.04</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.24, 0.18)</td>
<td>1.0e-01</td>
<td>0.22</td>
<td>62</td>
<td></td>
<td>0.04</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.79, 0.23)</td>
<td>5.0e-02</td>
<td>0.22</td>
<td>43</td>
<td></td>
<td>0.04</td>
<td>0.19</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>(-, 0.32)</td>
<td>1.3e-01</td>
<td>0.22</td>
<td>29</td>
<td></td>
<td>0.04</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.34, 0.05)</td>
<td>2.0e-02</td>
<td>0.56</td>
<td>195</td>
<td></td>
<td>0.04</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.19, 0.18)</td>
<td>8.0e-02</td>
<td>0.22</td>
<td>57</td>
<td></td>
<td>0.04</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.71, 0.22)</td>
<td>1.3e-01</td>
<td>0.22</td>
<td>40</td>
<td></td>
<td>0.04</td>
<td>0.19</td>
</tr>
</tbody>
</table>
### Table A.2: CEval of $\nu'_k$ with $j \in [3 - 8]$ and $k \in \{-, 0, 5, 10\}$, for Kubernetes codebase.

<table>
<thead>
<tr>
<th>$j$</th>
<th>$k$</th>
<th>D2F $\gamma$ diff and KS</th>
<th>D2D $\gamma$ diff</th>
<th>D2D KS</th>
<th>D2D max diff</th>
<th>newDeveloper untilT KS</th>
<th>$U_{\text{global} \text{dist}}$ KS</th>
<th>$U_{\text{local} \text{dist}}$ KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-</td>
<td>(-, 0.41) 5.9e-01 0.17 53</td>
<td></td>
<td></td>
<td></td>
<td>0.29</td>
<td>0.16</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(1.10, 0.17) 2.46e-01 0.56 219</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.41, 0.28) 5.8e-01 0.17 57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>(-, 0.41) 6.3e-01 0.17 31</td>
<td></td>
<td></td>
<td></td>
<td>0.23</td>
<td>0.09</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(1.10, 0.17) 3.9e-01 0.6 257</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.17, 0.26) 6.1e-01 0.17 57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>(-, 0.41) 6.0e-01 0.2 18</td>
<td></td>
<td></td>
<td></td>
<td>0.17</td>
<td>0.05</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(1.10, 0.17) 2.6e-01 0.56 85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.43, 0.26) 5.5e-01 0.16 24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>(-, 0.41) 8.7e-01 0.17 35</td>
<td></td>
<td></td>
<td></td>
<td>0.12</td>
<td>0.07</td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(1.10, 0.17) 6.4e-01 0.53 462</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.37, 0.26) 7.6e-01 0.14 78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>(-, 0.41) 7.2e-01 0.23 100</td>
<td></td>
<td></td>
<td></td>
<td>0.11</td>
<td>0.28</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(1.10, 0.17) 6.0e-01 0.67 639</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.39, 0.26) 7.1e-01 0.14 166</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>(-, 0.41) 6.6e-01 0.22 102</td>
<td></td>
<td></td>
<td></td>
<td>0.1</td>
<td>0.15</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(1.10, 0.17) 5.3e-01 0.56 352</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.48, 0.26) 6.3e-01 0.14 112</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>(0.05, 0.32) 6.5e-01 0.17 107</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table A.3: CEval of $s'_k$, with $j \in [3 - 8]$ and $k \in \{-, 0, 10, 15\}$, for Tensorflow codebase.

<table>
<thead>
<tr>
<th>$j$</th>
<th>$k$</th>
<th>$D2F \gamma_{diff}$ and $KS$</th>
<th>$FD2D \gamma_{diff}$</th>
<th>$FD2D \text{ max diff}$</th>
<th>$\text{newDeveloper until}T$</th>
<th>$U_{\text{global dist}}$</th>
<th>$U_{\text{local dist}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.17</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(-, 0.29)</td>
<td>2.1e-01</td>
<td>0.13</td>
<td>139</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>(0.78, 0.20)</td>
<td>4.6e-01</td>
<td>0.68</td>
<td>759</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>(0.33, 0.18)</td>
<td>2.2e-01</td>
<td>0.15</td>
<td>149</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(-, 0.29)</td>
<td>2.2e-01</td>
<td>0.13</td>
<td>219</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>(0.85, 0.21)</td>
<td>2.6e-01</td>
<td>0.67</td>
<td>647</td>
<td>0.21</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>(0.32, 0.18)</td>
<td>2.3e-01</td>
<td>0.14</td>
<td>241</td>
<td></td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>(0.76, 0.21)</td>
<td>2.7e-01</td>
<td>0.15</td>
<td>135</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>(0.81, 0.21)</td>
<td>5.9e-01</td>
<td>0.65</td>
<td>178</td>
<td>0.21</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>(0.22, 0.18)</td>
<td>1.9e-01</td>
<td>0.13</td>
<td>80</td>
<td></td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>(0.48, 0.21)</td>
<td>3.0e-01</td>
<td>0.13</td>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>(-, 0.29)</td>
<td>2.0e-02</td>
<td>0.13</td>
<td>107</td>
<td>0.24</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>(0.83, 0.21)</td>
<td>2.1e-01</td>
<td>0.61</td>
<td>1457</td>
<td></td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>(0.78, 0.21)</td>
<td>1.0e-02</td>
<td>0.13</td>
<td>219</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>(-, 0.29)</td>
<td>1.2e-01</td>
<td>0.14</td>
<td>336</td>
<td>0.25</td>
<td>0.32</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>(0.81, 0.21)</td>
<td>2.3e-01</td>
<td>0.73</td>
<td>2150</td>
<td></td>
<td>0.31</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>(0.34, 0.18)</td>
<td>1.3e-01</td>
<td>0.13</td>
<td>458</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>(-, 0.29)</td>
<td>1.4e-01</td>
<td>0.13</td>
<td>215</td>
<td>0.24</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>(0.82, 0.20)</td>
<td>2.3e-01</td>
<td>0.69</td>
<td>1134</td>
<td></td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>(0.60, 0.21)</td>
<td>4.3e-01</td>
<td>0.13</td>
<td>367</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table A.4: CEval of $s_{jk}$, with $j \in [3–8]$ and $k \in \{-, 0, 2.5, 5\}$, for Terraform codebase.

<table>
<thead>
<tr>
<th>$j$</th>
<th>$k$</th>
<th>D2F $\gamma$ diff and KS</th>
<th>FD2D $\gamma$ diff</th>
<th>FD2D KS</th>
<th>FD2D max diff</th>
<th>newDeveloper untilT KS</th>
<th>$U_{global,dist}$ KS</th>
<th>$U_{local,dist}$ KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-</td>
<td>(-, 0.31)</td>
<td>5.0e-02</td>
<td>0.24</td>
<td>33</td>
<td>0.19</td>
<td>0.15</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.74, 0.06)</td>
<td>6.3e-01</td>
<td>0.49</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.12, 0.13)</td>
<td>2.0e-01</td>
<td>0.24</td>
<td>71</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.42, 0.19)</td>
<td>7.0e-02</td>
<td>0.24</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>(-, 0.31)</td>
<td>3.0e-02</td>
<td>0.24</td>
<td>44</td>
<td>0.12</td>
<td>0.08</td>
<td>0.32</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.71, 0.06)</td>
<td>2.4e-01</td>
<td>0.52</td>
<td>152</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.18, 0.14)</td>
<td>4.0e-02</td>
<td>0.24</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.37, 0.20)</td>
<td>3.0e-02</td>
<td>0.24</td>
<td>54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>(-, 0.31)</td>
<td>1.1e-01</td>
<td>0.24</td>
<td>16</td>
<td>0.2</td>
<td>0.11</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.71, 0.06)</td>
<td>1.2e+00</td>
<td>0.45</td>
<td>73</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.03, 0.13)</td>
<td>3.0e-01</td>
<td>0.24</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.29, 0.20)</td>
<td>1.7e-01</td>
<td>0.24</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>(-, 0.31)</td>
<td>2.8e-01</td>
<td>0.24</td>
<td>40</td>
<td>0.07</td>
<td>0.13</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.75, 0.06)</td>
<td>5.0e-02</td>
<td>0.43</td>
<td>267</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.16, 0.13)</td>
<td>1.7e-01</td>
<td>0.24</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.25, 0.19)</td>
<td>1.7e-01</td>
<td>0.24</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>(-, 0.31)</td>
<td>7.0e-02</td>
<td>0.24</td>
<td>98</td>
<td>0.07</td>
<td>0.17</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.68, 0.06)</td>
<td>1.0e-02</td>
<td>0.5</td>
<td>417</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.08, 0.14)</td>
<td>1.0e-02</td>
<td>0.24</td>
<td>164</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.93, 0.20)</td>
<td>6.0e-02</td>
<td>0.24</td>
<td>118</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>(-, 0.31)</td>
<td>1.0e-02</td>
<td>0.24</td>
<td>89</td>
<td>0.06</td>
<td>0.07</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>(0.70, 0.06)</td>
<td>1.2e-01</td>
<td>0.48</td>
<td>190</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>(0.06, 0.14)</td>
<td>1.0e-02</td>
<td>0.24</td>
<td>105</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>(0.31, 0.20)</td>
<td>5.0e-02</td>
<td>0.24</td>
<td>97</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A.4 Third Iteration

The results for the third iteration, considering the representation of $s^j_k$, with $j = j_{rcb}$ and $k \in h_{rcb}$, per Dist, alongside the distinct evaluations are represented in Figures A.9 to A.12, respectively, for each RCB.

Considering Ansible, the best hypothesis is $h_{2.5}$, as across the different Dist, it achieves the non-trivial properties similar to the empirical Dist while having $\gamma_t \sim \gamma_e$ and small KS. In Dist = D2F/F/dist, D2F/F/dist resulting from $s^2_5$ has more similarities with the empirical, in terms of $\gamma$ and KS, while considering Dist = F2F/dist, F2F/dist resulting from $s^8_5$ has a more similar $\gamma$ with respect with the empirical F2F/dist. For Dist = FD2D/dist, DSA of FD2D/dist resulting from $s^2_5$ is highly similar with FD2D/dist resulting from $s^8_5$, however, the former has the most similar results with empirical FD2D/dist, visually and in terms of $\gamma$. Given this, we accept $s_{Dist}^8$ together with $h_{2.5}$ as a respectable set of inputs for Listing 6.3 to achieve results similar to the ones shown for the Ansible codebase.

For Kubernetes, the distinguishable hypothesis is $h_5$, given that: (1) even if $s^8_{10}$ reaches closer $\gamma$ for Dist = D2F/F/dist in relation with the empirical D2F/F/dist, $s^8_5$ has closer KS; (2) with respect to Dist = F2F/dist, the F2F/dist resulting from $s^8_5$ has more similarities with the $\gamma$ of the empirical F2F/dist; (3) for Dist = FD2D/dist, $s^8_{10}$ and $s^8_5$ have the most similar results with the empirical FD2D/dist, however, the FD2D/dist resulting from $s^8_5$ has small advantage. As already discussed, $\gamma_t$ is quite different from $\gamma_e$, given that the empirical FD2D/dist is characterized by a double power law while this does not happen for the FD2D/dist resulting from $s^8_5$. Visually, they have some similarities, therefore we admit $s_{Dist}^8$ together with $h_5$ as a sufficient set of inputs for Listing 6.3 to achieve results similar to the ones shown for the Kubernetes codebase.

Regarding Tensorflow, the preferable hypothesis is $h_{10}$. For Dist $\in \{D2F/F/dist, F2F/dist, FD2D/dist\}$, simulation $s^7_{10}$ leads to distributions with the most similar $\gamma_t$ when comparing with $\gamma_e$ and small KS. Given this, $s_{Dist}^8$ together with $h_{10}$ is the best set of inputs to achieve akin results to the ones observed for the Tensorflow codebase.

Finally, for Terraform, $h_{2.5}$ is the best hypothesis, as in the different Dist $\in \{D2F/F/dist, F2F/dist, FD2D/dist\}$, the Dist resulting from $s^2_{2.5}$ has the most similar $\gamma_t$ in comparison with $\gamma_e$ and smallest KS comparing with the other hypothesis. Then, $s_{Dist}^8$ plus $h_{2.5}$ is a sufficient set of inputs to produce similar results to the ones discussed for Terraform codebase.

In summary, the sufficient inputs to simulate each codebase are: (1) $s^8_5$ for Ansible; (2) $s^8_5$ for Kubernetes; (3) $s^7_{10}$ for Tensorflow; (4) $s^2_{2.5}$ for Terraform. We define $ss_{rcb}$ as sufficient inputs to simulate RCB.
(a) Representation of D2F/F/dist resulting from simulations $s^k_n$, with $k \in \{-, 0, 2.5, 5\}$ in comparison with empirical D2F/F/dist.

(b) Evals $(s^k_n, D2F/F/dist)$ with $k \in \{-, 0, 2.5, 5\}$.

(c) Representation of F2F/dist resulting from simulations $s^k_n$, with $k \in \{-, 0, 2.5, 5\}$ in comparison with empirical F2F/dist.

(d) Evals $(s^k_n, F2F/dist)$ with $k \in \{-, 0, 2.5, 5\}$.

(e) Representation of FD2D/dist resulting from simulations $s^k_n$, with $k \in \{-, 0, 2.5, 5\}$ in comparison with empirical FD2D/dist.

(f) Evals $(s^k_n, FD2D/dist)$ with $k \in \{-, 0, 2.5, 5\}$.

Figure A.9: Representation and evaluation of the resulting Dist from simulations $s^k_n$, with $k \in \{-, 0, 2.5, 5\}$, considering the Ansible codebase.
File D2F WD, wd
(number of changes in a file)

<table>
<thead>
<tr>
<th>k</th>
<th>$R_k$</th>
<th>$p_k$</th>
<th>KS</th>
<th>$\gamma_t$</th>
<th>$\gamma_e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>0.75</td>
<td>0.41</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>517</td>
<td>0.0</td>
<td>0.17</td>
<td>2.03</td>
<td>3.17</td>
</tr>
<tr>
<td>5</td>
<td>335</td>
<td>0.0</td>
<td>0.26</td>
<td>2.69</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>32</td>
<td>0.01</td>
<td>0.32</td>
<td>3.31</td>
<td></td>
</tr>
</tbody>
</table>

Cumulative distribution, $PCum_{wd}$

(empirical $k = -$, $k = 0.0$, $k = 5.0$, $k = 10.0$)

(a) Representation of D2F/dist resulting from simulations $s^k_{8}$, with $k \in \{-, 0, 5, 10\}$ in comparison with empirical D2F/dist.

File F2F WD, wd
(sum of number of files in commits where a certain file is present)

<table>
<thead>
<tr>
<th>k</th>
<th>$R_k$</th>
<th>$p_k$</th>
<th>KS</th>
<th>$\gamma_t$</th>
<th>$\gamma_e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-2</td>
<td>0.18</td>
<td>0.35</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>515</td>
<td>0.0</td>
<td>0.53</td>
<td>2.03</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>338</td>
<td>0.0</td>
<td>0.4</td>
<td>2.68</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>49</td>
<td>0.0</td>
<td>0.37</td>
<td>3.28</td>
<td></td>
</tr>
</tbody>
</table>

Cumulative distribution, $PCum_{wd}$

(empirical $k = -$, $k = 0.0$, $k = 5.0$, $k = 10.0$)

(b) Evals ($s^k_{8}$, D2F/dist) with $k \in \{-, 0, 5, 10\}$.

File FD2D WD, wd
(number of times that developer d have collaborated with other developers, in files)

<table>
<thead>
<tr>
<th>k</th>
<th>$R_k$</th>
<th>$p_k$</th>
<th>KS</th>
<th>$\gamma_t$</th>
<th>$\gamma_e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>498</td>
<td>0.0</td>
<td>0.22</td>
<td>1.8</td>
<td>2.46</td>
</tr>
<tr>
<td>0</td>
<td>278</td>
<td>0.0</td>
<td>0.56</td>
<td>1.93</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>579</td>
<td>0.0</td>
<td>0.14</td>
<td>1.83</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>396</td>
<td>0.0</td>
<td>0.17</td>
<td>1.81</td>
<td></td>
</tr>
</tbody>
</table>

Cumulative distribution, $PCum_{wd}$

(empirical $k = -$, $k = 0.0$, $k = 5.0$, $k = 10.0$)

(e) Representation of FD2D/dist resulting from simulations $s^k_{8}$, with $k \in \{-, 0, 5, 10\}$ in comparison with empirical FD2D/dist.

Figure A.10: Representation and evaluation of the resulting Dist from simulations $s^k_{8}$, with $k \in \{-, 0, 5, 10\}$, considering the Kubernetes codebase.
Figure A.11: Representation and evaluation of the resulting Dist from simulations $s_k^i$, with $k \in \{-, 0, 10, 15\}$, considering the Tensorflow codebase.
Figure A.12: Representation and evaluation of the resulting Dist from simulations $s_k^a$, with $k \in \{-0, 2.5, 5\}$, considering the Terraform codebase.
A.5 Collaboration per File Distributions

To show that $h_-$ is not sufficient to explain the collaboration process, we compare CollabPerFile/dist resulting from $s^j_k$, with $j = j_{rcb}$ and $k \in h_{rcb}$, per RCB. This is shown in Figure A.13, for each RCB. We can observe, per RCB, that CollabPerFile/dist resulting from $s^j_-$ does not have the non-trivial properties that the empirical CollabPerFile/dist has, while CollabPerFile/dist resulting from $ss_{rcb}$ has those non-trivial properties and show some similarities in terms of power law exponent.

![Graphs showing Cumulative distribution of collaboration per file](image)

**Figure A.13:** Representation of CollabPerFile/dist resulting from simulations $s^j_k$, with $j = j_{rcb}$, and $k \in h_{rcb}$ in comparison with empirical CollabPerFile/dist, across different analysed codebases.
A.6 Codebase Aspects Evaluation

To confirm $SS_{rcb}$ per RCB, we investigate CBAspEval between the theoretical timeline resultant from the algorithm described in Listing 6.1 and the empirical timelines. The results can be seen in Figure A.14, respectively, for each RCB. For Ansible, the majority of the aspects have small KS, however $filesAddedUntilT$ has 0.32 and $filesDeletedUntilT$ has 0.36. Considering Kubernetes, all of the aspects have KS $\leq 0.15$. Regarding Tensorflow, the max KS is 0.21 for $newDevelopersUntilT$. Finally, for Terraform, all aspects have small KS except $filesDeletedUntilT$ and $timePassedUntilT$. High KS means that the used distributions are not sufficient to explain both these aspects. Given that this is not the main focus of our study, we leave the analysis of these particular aspects and the impact that they may have on our simulation as an open problem.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitsPerDevDist</td>
<td>0.04</td>
</tr>
<tr>
<td>newDevelopersUntilT</td>
<td>0.03</td>
</tr>
<tr>
<td>$U_{global}$dist</td>
<td>0.10</td>
</tr>
<tr>
<td>$U_{local}$dist</td>
<td>0.09</td>
</tr>
<tr>
<td>$filesAddedUntilT$</td>
<td>0.32</td>
</tr>
<tr>
<td>$filesChangedUntilT$</td>
<td>0.12</td>
</tr>
<tr>
<td>$filesDeletedUntilT$</td>
<td>0.36</td>
</tr>
<tr>
<td>$timePassedUntilT$</td>
<td>0.17</td>
</tr>
</tbody>
</table>

(a) Ansible.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitsPerDevDist</td>
<td>0.13</td>
</tr>
<tr>
<td>newDevelopersUntilT</td>
<td>0.10</td>
</tr>
<tr>
<td>$U_{global}$dist</td>
<td>0.15</td>
</tr>
<tr>
<td>$U_{local}$dist</td>
<td>0.06</td>
</tr>
<tr>
<td>$filesAddedUntilT$</td>
<td>0.10</td>
</tr>
<tr>
<td>$filesChangedUntilT$</td>
<td>0.04</td>
</tr>
<tr>
<td>$filesDeletedUntilT$</td>
<td>0.07</td>
</tr>
<tr>
<td>$timePassedUntilT$</td>
<td>0.12</td>
</tr>
</tbody>
</table>

(b) Kubernetes.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitsPerDevDist</td>
<td>0.08</td>
</tr>
<tr>
<td>newDevelopersUntilT</td>
<td>0.21</td>
</tr>
<tr>
<td>$U_{global}$dist</td>
<td>0.10</td>
</tr>
<tr>
<td>$U_{local}$dist</td>
<td>0.18</td>
</tr>
<tr>
<td>$filesAddedUntilT$</td>
<td>0.13</td>
</tr>
<tr>
<td>$filesChangedUntilT$</td>
<td>0.01</td>
</tr>
<tr>
<td>$filesDeletedUntilT$</td>
<td>0.11</td>
</tr>
<tr>
<td>$timePassedUntilT$</td>
<td>0.16</td>
</tr>
</tbody>
</table>

(c) Tensorflow.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>commitsPerDevDist</td>
<td>0.08</td>
</tr>
<tr>
<td>newDevelopersUntilT</td>
<td>0.06</td>
</tr>
<tr>
<td>$U_{global}$dist</td>
<td>0.07</td>
</tr>
<tr>
<td>$U_{local}$dist</td>
<td>0.13</td>
</tr>
<tr>
<td>$filesAddedUntilT$</td>
<td>0.08</td>
</tr>
<tr>
<td>$filesChangedUntilT$</td>
<td>0.05</td>
</tr>
<tr>
<td>$filesDeletedUntilT$</td>
<td>0.25</td>
</tr>
<tr>
<td>$timePassedUntilT$</td>
<td>0.28</td>
</tr>
</tbody>
</table>

(d) Terraform.

Figure A.14: CBAspEval, per RCB, between its empirical timeline and the theoretical timeline generated in Listing 6.1 with support from $sDists_{rcb}$. 