UNIVERSITY OF PISA
DEPARTMENT OF COMPUTER SCIENCE
DOTTORATO DI RICERCA (PhD) IN COMPUTER SCIENCE

PhD THESIS

Process Models for Learning Patterns in FLOSS Repositories

Patrick Mukala

SUPERVISOR
Prof. Franco Turini

SUPERVISOR
Dr. Antonio Cerone

ADVISORY COMMITTEE
Prof. Turini
Dr. Cerone
Prof. Pedreschi
Dr. Gervasi

2015
ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and provided valuable assistance in the preparation and completion of this research.

First and foremost, my utmost gratitude to the Almighty God: without Him, none of this would have been possible. I owe Him my life and everything that I will ever accomplish. His grace and guidance have been the rock for the continuation of this research.

It is because of the support, patience and guidance of the following people that the completion of this thesis has been possible. To them, I owe my deepest gratitude:

Professor Franco Turini and Dr. Antonio Cerone, who undertook to act as my supervisors, despite their many other academic and professional commitments; their expertise, vast knowledge and guidance were very significant in the realization of this study. Furthermore, my gratitude goes to all the members of the advisory committee, Professor Pedreschi and Dr. Gervasi for their invaluable input and feedback during the evaluations throughout the period of this thesis. I would also like to extend my gratitude to Professor Degano, the chair of the PhD school, for all his help and support during my time in the Computer Science department.

And finally, I am grateful for the undying support I have received throughout this period from my family. My wife has been my rock during this time; I thank her for leaving her job to be with me in Pisa; my parents, Corneille Mulamba and Esther Ngalula, for their steadfast encouragement while I was completing this research.

The research work carried out in this thesis started within the UNU-IIST project Peer-Production Approaches to e-Learning (PPAeL), led by Dr. Antonio Cerone and supported by Macao Science and Technology Development Fund, File No. 019/2011/A1. A scholarship from the United Nations University supported me during my PhD time in Pisa.
This thesis is dedicated to my princess, my daughter:
Eliore-Valerie
ABSTRACT

Evidence suggests that Free/Libre Open Source Software (FLOSS) environments provide unlimited learning opportunities. Community members engage in a number of activities both during their interaction with their peers and while making use of these environments’ repositories. To date, numerous studies document the existence of learning processes in FLOSS through surveys or by means of questionnaires filled by FLOSS projects participants. At the same time, there is a surge in developing tools and techniques for extracting and analyzing data from different FLOSS data sources that has birthed a new field called Mining Software Repositories (MSR).

In spite of these growing tools and techniques for mining FLOSS repositories, there is limited or no existing approaches to providing empirical evidence of learning processes directly from these repositories. Therefore, in this work we sought to trigger such an initiative by proposing an approach based on Process Mining. With this technique, we aim to trace learning behaviors from FLOSS participants’ trails of activities as recorded in FLOSS repositories. We identify the participants as Novices and Experts. A Novice is defined as any FLOSS member that benefits from a learning experience through acquiring new skills while the Expert is the provider of these skills.

The significance of our work is mainly twofold. First and foremost, we extend the MSR field by showing the potential of mining FLOSS repositories by applying Process Mining techniques. Lastly, our work provides critical evidence that boosts the understanding of learning behavior in FLOSS communities by analyzing the relevant repositories. In order to accomplish this, we have proposed and implemented a methodology that follows a seven-step approach including developing an appropriate terminology for learning processes in FLOSS, contextualizing learning processes through a-priori models, generating Event Logs, generating corresponding process models, interpreting and evaluating the value of process discovery, performing conformance analysis and verifying a number of formulated hypotheses with regard to tracing learning patterns in FLOSS communities.

The implementation of this approach has resulted in the development of the Ontology of Learning in FLOSS (OntoLiFLOSS) environments that defines the terms needed to describe learning processes in FLOSS as well as providing a visual representation of these processes through Petri net-like Workflow nets. Moreover, another novelty pertains to the mining of FLOSS repositories by defining and describing the preliminaries required for preprocessing FLOSS data before applying Process Mining techniques for analysis. Through a step-by-step process, we effectively detail how the Event Logs are constructed through generating key phrases and making use of Semantic Search.

Taking a FLOSS environment called Openstack as our data source, we apply our proposed techniques to identify learning activities based on key phrases catalogs and classification rules expressed through pseudo code as well as the appropriate Process Mining tool. We thus produced Event Logs that are based on the semantic content of messages in Openstack’s Mailing archives, Internet Relay Chat (IRC) messages, Reviews, Bug reports and Source code to retrieve the corresponding activities. Considering these repositories in light of the three learning process phases (Initiation, Progression and maturation), we produced an Event Log for each participant (Novice or Expert) in every phase on the corresponding dataset. Hence, we produced 14 Event Logs that helped build 14 corresponding process maps which are visual representation of the flow occurrence of learning activities in FLOSS for each participant.

These process maps provide critical indications that speak volumes in terms of the presence of learning processes in the analyzed repositories. The results show that learning activities do occur at a significant rate during messages exchange on both Mailing archives and IRC messages. The slight differences between the two datasets can be highlighted in two ways. First, the involvement of Experts is more on
IRC than it is on Mailing archives with 7.22% and 0.36% of Expert involvement respectively on IRC forums and Mailing lists. This can be justified by the differences in the length of messages sent on these two datasets. The average length of sent messages is 3261 characters for an email compared to 60 characters for a chat message. The evidence produced from this mining experiment solidifies the finding in terms of the existence of learning processes in FLOSS as well as the scale at which they occur. While the Initiation phase shows the Novice as the most involved in the start of the learning process, during Progression phase the involvement of the Expert can be seen to be significantly increasing.

In order to trace the advanced skills in the Maturation phase, we look at repositories that store data about developing, creating code, examining and reviewing the code, identifying and fixing possible bugs. Therefore, we consider three repositories including Source Code, Bug reports and Reviews. The results obtained in this phase largely justify the choice of these three datasets to track learning behavior at this stage. Both the Bug reports and the Source code demonstrate the commitment of the Novice to seek answers and interact as much as possible in strengthening the acquired skills. With a participation of 49.22% for the Novice against 46.72% for the Expert and 46.19% against 42.04% respectively on Bug reports and Source code, the Novice still engages significantly in learning. On the last dataset, Reviews, we notice an increase in the Expert’s role. The Expert performs activities to the tune of 40.36% of total number of activities against 22.17% for the Novice.

The last steps of our methodology steer the comparison of the defined a-priori models with final models that describe how learning processes occur according to the actual behavior from Event Logs. Our attempts to producing process models start with depicting process maps to track the actual behaviour as it occurs in Openstack repositories, before concluding with final Petri net models representative of learning processes in FLOSS as a result of conformance analysis.

For every dataset in the corresponding learning phase, we produce 3 process maps respectively depicting the overall learning behaviour for all FLOSS community members (Novice or Expert together), then the Novice and Expert. In total, we produced 21 process maps, empirically describing process models on real data, 14 process models in the form of Petri nets for every participant on each dataset.

We make use of the Artificial Immune System (AIS) algorithms to merge the 14 Event Logs that uniquely capture the behaviour of every participant on different datasets in the three phases. We then reanalyze the resulting logs in order to produce 6 global models that inclusively provide a comprehensive depiction of participants’ learning behavior in FLOSS communities. This description hints that Workflow nets introduced as our a-priori models give rather a more simplistic representation of learning processes in FLOSS. Nevertheless, our experiments with Event Logs starting from process discovery to conformance checking from Openstack repositories demonstrate that the real learning behaviors are more complete and most importantly largely submerge these simplistic a-priori models.

Finally, our methodology has proved to be effective in both providing a novel alternative for mining FLOSS repositories and providing empirical evidence that describes how knowledge is exchanged in FLOSS environments. Moreover, our results enrich the MSR field by providing a reproducible step-by-step problem solving approach that can be customized to answer subsequent research questions in FLOSS repositories using Process Mining.

**Keywords**— FLOSS learning processes, learning activities in Open Source, Mining Software Repositories, Process Mining, Semantic Search.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS .................................................................................................................... i
ABSTRACT ........................................................................................................................................ iii
TABLE OF CONTENTS ....................................................................................................................... v
LIST OF TABLES ............................................................................................................................... xvi

CHAPTER 1: INTRODUCTION ........................................................................................................... 1
  1.1 Introduction ................................................................................................................................ 1
    1.1.1 Background ........................................................................................................................... 1
    1.1.2 Motivation ............................................................................................................................. 1
  1.2 Description of Topics .................................................................................................................... 2
    1.2.1 FLOSS environments .......................................................................................................... 2
    1.2.1.1 Generic Overview ........................................................................................................... 2
    1.2.1.2 Participant Profiles .......................................................................................................... 3
    1.2.1.3 Roles and Responsibilities of FLOSS members ............................................................... 4
    1.2.1.4 Collaboration and Contribution activities of FLOSS members ........................................ 6
    1.2.2 FLOSS Communities as Learning Environments ................................................................... 9
    1.2.2.1 Learning and Activity Patterns in FLOSS ....................................................................... 9
    1.2.2.2 FLOSS as E-learning Tools ............................................................................................. 15
    1.2.2.3 Learning Awareness ......................................................................................................... 16
    1.2.2.4 Learning according to FLOSScom .................................................................................. 17
  1.3 Related Work .............................................................................................................................. 18
  1.4 Proposed Approach .................................................................................................................... 22
  1.5 Conclusion ................................................................................................................................. 28
  1.6 Structure of the Thesis ................................................................................................................ 28

CHAPTER 2: MINING SOFTWARE REPOSITORIES: STATE OF THE-ART ........................................ 30
  2.1 Introduction .................................................................................................................................. 30
  2.2 Mining Software Repositories: Leading factors ......................................................................... 30
  2.3 Mining Techniques: Selected, Relevant Approaches .................................................................... 31
  2.4 Tools .......................................................................................................................................... 39
  2.5 Conclusion: Process Mining and FLOSS repositories ................................................................. 41

CHAPTER 3: PROPOSED APPROACH ............................................................................................. 42
  3.1 Introduction .................................................................................................................................. 42
  3.2 Step 1: Ontology for Learning Processes in FLOSS (OntoLiFLOSS) ............................................. 42
    3.2.1 Methodology ....................................................................................................................... 43
    3.2.2 OntoLiFLOSS: Main Concepts ............................................................................................ 44
      3.2.2.1 Classes .......................................................................................................................... 44
3.2.2.2 Properties ........................................................................................................... 58

3.3 Step 2: Identification and Specification of Learning Processes ................................ 59

3.3.1 Workflow Petri nets for Learning Processes in FLOSS: Background and Preliminaries ........................................................................................................................................................................ 60

3.3.2 Phase 1: Initiation.................................................................................................. 62

3.3.3 Phase 2: Progression ......................................................................................... 65

3.3.4 Phase 3: Maturation ......................................................................................... 69

3.4 Conclusion ........................................................................................................... 73

CHAPTER 4: PROCESS MODELS FOR LEARNING PROCESS: PRELIMINARIES ............... 74

4.1 Introduction ............................................................................................................ 74

4.2 Process Models for Learning Process: Crucial landmarks ....................................... 75

4.2.1 Heuristics .......................................................................................................... 76

4.2.2 Mining data sources ......................................................................................... 77

4.2.3 Semantic Search with SQL ................................................................................ 78

4.3 Constructing Event Logs ...................................................................................... 79

4.3.1 Generation of key phrases ................................................................................ 80

4.3.1.1 Automatic assignment /discovery of key phrases ...................................... 80

4.3.1.2 Previous studies on users’ interactions in FLOSS communities .................. 82

4.4 Conclusion ........................................................................................................... 99

CHAPTER 5: PROCESS MODELS FOR LEARNING PROCESS: TECHNICAL SPECIFICATIONS ................................................................................................................................. 100

5.1 Introduction ............................................................................................................ 100

5.2 Data Source .......................................................................................................... 100

5.3 Data Analysis ........................................................................................................ 101

5.3.1 Enabling Semantic Search ................................................................................ 101

5.3.2 Rules for identification of learning Activities .................................................. 104

5.4 DISCO as a tool for Process Mining ..................................................................... 109

5.5 Conclusion ........................................................................................................... 110

CHAPTER 6: PROCESS MAPS FOR INITIATION PHASE: EMPIRICAL RESULTS ............... 111

6.1 Introduction ............................................................................................................ 111

6.2 Mailing Archives’ Messages .................................................................................. 111

6.3 Internet Relay Chats’ Messages ............................................................................ 130

6.4 Conclusion ........................................................................................................... 148

CHAPTER 7: PROCESS MAPS FOR PROGRESSION PHASE: EMPIRICAL RESULTS ........ 149

7.1 Introduction ............................................................................................................ 149

7.2 Mailing archives’ Messages ................................................................................. 149

7.3 Internet Relays Chat Messages .............................................................................. 166

7.4 Conclusion ........................................................................................................... 183

CHAPTER 8: PROCESS MAPS FOR MATURATION PHASE: EMPIRICAL RESULTS ............... 184

8.1 Introduction ........................................................................................................... 184
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FLOSS community members by profession [18]</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>An organizational structure of a typical FLOSS community</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Role transition in FLOSS communities [2]</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Individual Team interplay for FLOSS</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Methodology to Identify developers from multiple repositories</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Making tacit knowledge explicit in FLOSS communities</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>Learning process of individual actors in OSS communities</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>Learning Stages and contributor’s activities in OSS communities</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>Learning Stages and Activity contents</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>Example of a workflow process modeled as a Petri net</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>A Graphical representation of our proposed problem solving approach</td>
<td>24</td>
</tr>
<tr>
<td>12</td>
<td>Methodology for Learning Patterns Identification</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>Rigorous manual inspection of randomly selected links for Bug fixing [57]</td>
<td>33</td>
</tr>
<tr>
<td>14</td>
<td>Matching is based on prerequisite information [61]</td>
<td>36</td>
</tr>
<tr>
<td>15</td>
<td>Summary of steps for Developers’ identity matching in FLOSS [61]</td>
<td>37</td>
</tr>
<tr>
<td>16</td>
<td>Depiction of Hipikat’s main objects [66]</td>
<td>39</td>
</tr>
<tr>
<td>17</td>
<td>CVSgrab’s architectural pipeline [69]</td>
<td>40</td>
</tr>
<tr>
<td>18</td>
<td>SoftChange Architecture [70]</td>
<td>40</td>
</tr>
<tr>
<td>19</td>
<td>OntoLiFLOSS Main Classes</td>
<td>45</td>
</tr>
<tr>
<td>20</td>
<td>Graph for OntoLiFLOSS Main Classes</td>
<td>45</td>
</tr>
<tr>
<td>21</td>
<td>Network graph of the FLOSSProject Class, immediate subclasses and related classes</td>
<td>46</td>
</tr>
<tr>
<td>22</td>
<td>Network graph of the LearningActivity Class, immediate subclasses and related classes</td>
<td>47</td>
</tr>
<tr>
<td>23</td>
<td>Subclasses during Initiation Stage</td>
<td>47</td>
</tr>
<tr>
<td>24</td>
<td>Subclasses during Progression Stage</td>
<td>48</td>
</tr>
<tr>
<td>25</td>
<td>Subclasses during Maturation Stage</td>
<td>49</td>
</tr>
<tr>
<td>26</td>
<td>Network graph of the Resource Class, immediate subclasses and related classes</td>
<td>50</td>
</tr>
<tr>
<td>27</td>
<td>Sub classification of the Resource Class</td>
<td>51</td>
</tr>
<tr>
<td>28</td>
<td>Network graph of the Repository Class, immediate subclasses and related classes</td>
<td>52</td>
</tr>
<tr>
<td>29</td>
<td>Network graph of the Team Class and related classes</td>
<td>52</td>
</tr>
<tr>
<td>30</td>
<td>Network graph of the StandardOfPractice Class, immediate subclasses and related classes</td>
<td>53</td>
</tr>
<tr>
<td>31</td>
<td>Network graph of the Participant Class, immediate subclasses and related classes</td>
<td>53</td>
</tr>
<tr>
<td>32</td>
<td>Network graph of the Novice Class and related classes</td>
<td>54</td>
</tr>
<tr>
<td>33</td>
<td>Network graph of the Expert Class and related classes</td>
<td>54</td>
</tr>
<tr>
<td>34</td>
<td>Graph representation of instances of the LearningProcess Class</td>
<td>55</td>
</tr>
<tr>
<td>35</td>
<td>Network graph of the LearningProcess Class and related classes</td>
<td>55</td>
</tr>
<tr>
<td>36</td>
<td>Network graph of the ProjectRole Class, subclasses and related classes</td>
<td>56</td>
</tr>
<tr>
<td>37</td>
<td>Network graph of the LearningStage Class, subclasses and related classes</td>
<td>57</td>
</tr>
<tr>
<td>38</td>
<td>Workflow net for Novice in Initiation phase</td>
<td>63</td>
</tr>
<tr>
<td>39</td>
<td>Workflow net for Expert in Initiation phase</td>
<td>64</td>
</tr>
<tr>
<td>40</td>
<td>Workflow net for Novice in Progression phase</td>
<td>67</td>
</tr>
<tr>
<td>41</td>
<td>Workflow net for Expert in Progression phase</td>
<td>68</td>
</tr>
<tr>
<td>42</td>
<td>Workflow net for Novice in Maturation phase</td>
<td>71</td>
</tr>
<tr>
<td>43</td>
<td>Workflow net for Expert in Maturation phase</td>
<td>72</td>
</tr>
<tr>
<td>44</td>
<td>Process Mining Meta Data Model</td>
<td>74</td>
</tr>
</tbody>
</table>
Figure 237. Second Group of Activities depicting second point of iteration
Figure 238. Third Group of Activities depicting third point of iteration
Figure 239. Final Learning Model for Novice during Initiation phase on IRC with a replay of 1000 traces
Figure 240. First Group of Activities depicting first point of iteration
Figure 241. Second Group of Activities depicting second point of iteration
Figure 242. Diagnostics Model appropriateness on both Mailing archives and IRC messages for Expert in FLOSS
Figure 243. Final Learning Model for Expert during Initiation phase on Mailing archives with simulated behavior using 1000 traces
Figure 244. Final Learning Model for Expert during Initiation phase on IRC with a replay of 1000 traces
Figure 245. First Group of Activities depicting first point of iteration
Figure 246. Second Group of Activities depicting second point of iteration
Figure 247. Diagnostics Model appropriateness on Mailing archives for Novice in FLOSS during Progression Phase
Figure 248. Diagnostics Model appropriateness on Internet Relay Chat messages for Novice in FLOSS during Progression Phase
Figure 249. Final Learning Model for Novice during Progression phase on Mailing archives with a replay of 1000 traces
Figure 250. First Group of Activities depicting first point of iteration
Figure 251. Second Group of Activities depicting second point of iteration
Figure 252. Final Learning Model for Novice during Progression phase on IRC with a replay of 1000 traces
Figure 253. First Group of Activities depicting first point of iteration
Figure 254. Second Group of activities depicting second point of iterations
Figure 255. Diagnostics Model appropriateness on Mailing archives for Expert in FLOSS during Progression phase
Figure 256. Diagnostics Model appropriateness on Internet relay Chat messages for Expert in FLOSS during Progression phase
Figure 257. Final Learning Model for Expert during Progression phase on Mailing with a replay of 1000 traces
Figure 258. First Group of activities depicting first point of iterations
Figure 259. Second Group of activities depicting second point of iterations
Figure 260. Final Learning Model for Expert during Progression phase on IRC with a replay of 1000 traces
Figure 261. First Group of activities depicting first point of iterations
Figure 262. Second and Third Group of activities depicting second point of iterations
Figure 263. Diagnostics Model appropriateness on Bug Reports for Novice in FLOSS during Maturation phase
Figure 264. Diagnostics Model appropriateness on Reviews for Novice in FLOSS during Maturation phase
Figure 265. Diagnostics Model appropriateness on Source code for Novice in FLOSS during Maturation phase
Figure 266. Final Learning Model for Novice during Maturation phase on Bug reports with a replay of 1000 traces
Figure 267. First Grouping of Activities indicating first point of iteration
Figure 268. Second Grouping of Activities indicating second point of iteration
Figure 269. Final Learning Model for Novice during Maturation phase on Reviews with a replay of 1000 traces
Figure 270. First grouping of activities depicting possible starting points of paths
Figure 271. Second grouping of activities depicting possible points of iterations
Figure 272. Third grouping of activities depicting possible points of iterations
Figure 273. Final Learning Model for Novice during Maturation phase on Source Code with a replay of 1000 traces
Figure 274. First grouping of activities depicting possible points of iterations and beginning of paths
Figure 275. Second grouping of activities depicting remaining possible points of iterations and traces paths
Figure 276. Diagnostics Model appropriateness on Bug reports for Expert in FLOSS during Maturation phase
Figure 277. Diagnostics Model appropriateness on Reviews for Expert in FLOSS during Maturation phase

Figure 278. Diagnostics Model appropriateness on Source Code for Expert in FLOSS during Maturation phase

Figure 279. Final Learning Model for Expert during Maturation phase on Bug reports with a replay of 1000 traces

Figure 280. First grouping of activities depicting possible starting point of traces and iteration points

Figure 281. Second grouping of activities depicting possible second points of iteration

Figure 282. Final Learning Model for Expert during Maturation phase on Reviews with a replay of 1000 traces

Figure 283. First grouping of activities showing possible starting points for distinct traces and points of iterations

Figure 284. Second grouping of activities showing possible points of iterations and continuation of paths

Figure 285. Final Learning Model for Expert during Maturation phase on Source Code with a replay of 1000 traces

Figure 286. First grouping of activities showing possible starting points for distinct traces and points of iterations

Figure 287. Second grouping of activities showing possible points of iterations and continuation of paths

Figure 288. FINAL PROCESS MODEL for NOVICE from merged Event Logs

Figure 289. First Grouping of activities explaining iterations and starting points of traces

Figure 290. Second Grouping of activities explaining iterations and ending points of traces

Figure 291. INITIAL APRIORI MODEL for NOVICE with a replay of 1000 traces

Figure 292. Grouping of activities explaining alternative activities in the traces

Figure 293. FINAL PROCESS MODEL for EXPERT from merged Event Logs

Figure 294. First Grouping of activities explaining iterations and starting points of traces for Expert

Figure 295. Second Grouping of activities explaining iterations and ending points of traces for Expert

Figure 296. INITIAL APRIORI MODEL for EXPERT with a replay of 1000 traces

Figure 297. Grouping of activities explaining alternative activities in the traces for the EXPERT

Figure 298. FINAL PROCESS MODEL for NOVICE from merged Event Logs during Progression

Figure 299. First Grouping of activities explaining variation points for Novice during Progression

Figure 300. Second Grouping of activities explaining variation points for Novice during progression

Figure 301. INITIAL APRIORI MODEL for NOVICE during Progression with a replay of 1000 traces

Figure 302. Grouping of activities on the A-PRIORI MODEL depicting variations for NOVICE with a replay of 1000 traces

Figure 303. FINAL PROCESS MODEL for EXPERT from merged Event Logs during Progression

Figure 304. First Grouping of activities explaining variation points for Expert during progression

Figure 305. Second Grouping of activities explaining variation points for Expert during Progression

Figure 306. Third Grouping of activities explaining variation points for Expert during progression

Figure 307. INITIAL APRIORI MODEL for EXPERT during Progression with a replay of 1000 traces

Figure 308. First grouping of activities on the A-PRIORI MODEL depicting variations for EXPERT with a replay of 1000 traces

Figure 309. Second grouping of activities on the A-PRIORI MODEL depicting variations for EXPERT with a replay of 1000 traces

Figure 310. FINAL PROCESS MODEL for NOVICE from merged Event Logs during Maturation

Figure 311. First Grouping of activities explaining variation points for Novice during Maturation

Figure 312. Second Grouping of activities explaining variation points for Novice during Maturation

Figure 313. INITIAL APRIORI MODEL for NOVICE during Maturation with a replay of 1000 traces

Figure 314. First grouping of activities on the A-PRIORI MODEL depicting variations for NOVICE with a replay of 1000 traces

Figure 315. Second grouping of activities on the A-PRIORI MODEL depicting variations for NOVICE with a replay of 1000 traces

Figure 316. Third grouping of activities on the A-PRIORI MODEL depicting variations for NOVICE with a replay of 1000 traces
LIST OF TABLES

Table 1. Age structure of the FLOSS community, 1991 to 2002 [18] ................................................................. 3
Table 2. A workflow log ........................................................................................................................................ 21
Table 3. Relationships depicting Associations between classes in OntoLiFLOSS ........................................... 58
Table 4. Initial assumptions for Process Mining learning activities from FLOSS repositories ......................... 76
Table 5. Key attributes per table and database for Semantic Search ............................................................... 79
Table 6. Examples of Key phrases in a Q&A thread from an Apache forum [114] ............................................. 92
Table 7. CATALOG OF KEY PHRASES FOR LEARNING ACTIVITIES AT INTIATION ................................. 94
Table 8. CATALOG OF KEY PHRASES FOR LEARNING ACTIVITIES AT PROGRESSION PHASE .......... 95
Table 9. CATALOG OF KEY PHRASES FOR LEARNING ACTIVITIES AT MATURATION PHASE .............. 96
Table 10. Mailing Archives Description ........................................................................................................ 112
Table 11. Internet relay Chat Messages Description ......................................................................................... 130
Table 12. Bug reports dataset Description ....................................................................................................... 185
Table 13. Reviews dataset Description ............................................................................................................ 203
Table 14. Source Code dataset Description ..................................................................................................... 221
Table 15. Conformance Analysis Results in FLOSS Repositories .................................................................. 257
Table 16. Projection of Unique Traces in both A-Priori and Final Models .......................................................... 335
Table 17. A Summary of Differences in Number of Unique Traces between A-Priori and Final Models for Novice ........................................................................................................................................ 342
Table 18. A Summary of Differences in Number of Unique Traces between A-Priori and Final Models for Expert ........................................................................................................................................ 342
CHAPTER 1: INTRODUCTION

1.1 Introduction

1.1.1 Background

Free/Libre Open Source Software (FLOSS) projects are considered an example of Commons-Based Peer-Production models where groups of participants work remotely and usually in the absence of traditional hierarchical structures to achieve projects of common purposes [7-8]. While the phenomenon of FLOSS projects has generated considerable research interest [1-6], it still offers extensive potential worth exploring. In particular, in the context of learning, FLOSS communities have been established as environments where successful collaborative and participatory learning between participants occurs [6-10] while working on software projects. Many successful projects have produced applications of widespread popularity. Such OSS products include operating systems such as Linux, network services such as Apache, high-end applications such as MySQL, and Learning Management Systems (LMS), such as Moodle [11]. The quality of these products is indicative of their widespread use and popularity.

This has resulted in growing research interests on FLOSS communities including exploring them as learning environments. While many studies have provided invaluable insights in this direction, their results are mostly based on surveys and observation reports [13]. Given the evidence that learning does surely occur within this context as indicated by numerous studies, little has been done in mining these repositories for learning patterns. Therefore, this work proposes to contribute in this context by studying learning patterns from FLOSS repositories with the hope to provide insights in this context based on empirical evidence using Process Mining.

In this chapter, we set the tone for our research endeavor by contextualizing the topic and the motivation thereof; defining the scope as well as the methodology we propose to follow in order to produce results.

1.1.2 Motivation

This work is driven by the need to foster the understanding of learning patterns generated by knowledge acquisition mechanisms occurring in FLOSS environments. As evidence of consideration for FLOSS environments as teaching tools even in formal education setting [6, 10] mounts, we think it is still fit to provide more empirical evidence to help study learning behaviors. These behaviors can be represented in forms of processes that can highlight the way in which learning occurs. This can be achieved through Process Mining.

Process Mining helps discover processes from Event Logs and based on these processes (activities or messages being exchanged between developers), a process model is produced [14]. Since the mid-nineties, considerable work in the field of Process Mining has been conducted to develop techniques, design and implement algorithms for process models from Event Logs in Business Process Management (BPM) [12]. The initial success of Process Mining algorithms demonstrates their potential in applying them in software process modeling. One such example is the use of a Process Mining framework to help model software development processes by making use of Software Configuration management (SCM) systems such as CVS or subversion systems [13]. This work exemplifies how Process Mining can be applied to understand software development processes based on audit trail documents recorded by the SCM during the development cycle. Another example documents the use of such algorithms in combining data from numerous software repositories in order to generate a log for Process Mining and analysis [15].
It is critical to note that using Process Mining presupposes the availability of an Event Log in a suitable format. Hence, our key motivation is to define and implement an effective approach to constructing Event Logs from which Process Mining can identify learning behaviors and patterns, expressed in process models. These models can serve as analytical models for learning patterns in FLOSS environments.

In the rest of the chapter, we briefly discuss the concepts as mentioned above and contextualize the scope of our research.

1.2 Description of Topics

1.2.1 FLOSS environments

1.2.1.1 Generic Overview
Free/Libre Open Source Software (FLOSS) environments are online communities made up of heterogeneous participants who remotely interact in the development of Open Source Software. The fundamental idea behind these environments is to enable free access to software source code to users whenever they wish to [1-2]. Software .exe files or binaries are freely available for download by users under the guidance of the General Public License (GPL). This gives the users the freedom to modify the source code, incorporate it in new software applications or anything they wish to achieve as long as these products would carry the same license [176].

Given the aura around the phenomenon of FLOSS, there exist a number of different terms used to describe the concept depending on the context use of the products and the related license [176]. Some of these terms include Free Software (FS), as used by Free Software Foundation (FSF), Libre Software (LS), Open Source Software (OSS) used by the Open Source Initiative (OSI), Free Open Source Software (FOSS), and Free/Libre/Open Source Software (FLOSS) are different terms used in reference with the concept of free software [1-2].

The term “FLOSS” as used in this document and the rest of the research, borrows its basic definition from the FLOSScom Project [1-2]. In this work, it refers to users’ freedom to use, modify, distribute, or even sell the software with little obligations as in propriety or closed source software”. This implies that when users have access to the software, they can decide to make their contribution public to the FLOSS community or just keep it private. In FLOSS environments, developers/participants make use of online platforms such as Sourceforge, Freshmeat or GitHub for all software development activities. Some common repositories and tools used include de facto versioning systems such as Concurrent Versions System (CVS) or Subversion (SVN), bug-tracking systems (BTS) and bug databases (Bugzilla). These repositories also make provision for different ways in which participants communicate and interact such as mailing lists, wikis, forums and Internet Relay Chats (IRC) [1].

The current state of affairs demonstrates that with the expansion of the Internet, FLOSS has drastically changed software development and distribution [17]. With FLOSS communities, the internet enables a large number of remotely distributed individuals who work together towards software projects in a Bazaar model-like fashion [17]. These participants, through extensive collaboration, contribute in writing code, debugging, testing and integrating applications as well as any other required Software Engineering (SE) activities. Other activities in the communities range from support services such as product features suggestions, distribution, query handling, new members induction[1-2]. The development process in FLOSS communities is an example of a viable and alternative approach to software development partly due to the quality of software produced [1-2].
In recent years, FLOSS environments have been credited to deliver high-quality software products such as the Linux kernel, the Apache HTP server, Sendmail, MySQL, PostgreSQL and Moodle among others [1, 11].

Numerous studies have recently analyzed FLOSS environments and projects to determine the quality of OSS products. Specifically, these studies analyzed the activities, best practices and collaboration patterns within FLOSS environments in order to identify factors that enable the emergence of high quality OSS products. Making use of a set of quality metrics, Halloran and Scherlis [16] reviewed a number of FLOSS projects in order to reveal some prospects on the link between high quality and open source software practices. This study highlighted good project communication and management as key enablers. Additional studies have been conducted by Coverity, a prominent software defects detection company, using Coverity Scan, a tool for automated static analysis of source code [11]. Coverity reports accentuate the high-quality of OSS products, showing that it is comparable to or even better than source proprietary software [11].

1.2.1.2 Participant Profiles

In order to study and explore FLOSS communities, we look at some data pertaining to personal features of the participants. These characteristics include age, qualification and professional background etc. The FLOSS Developers Survey [18] shows that the age of Open Source Software communities is indicative of a great interest in online collaborative software development. The survey indicates that individuals who become part of these communities are aged between 14 and 73 years old with a predominance of participants aged between 16 and 36 years [1] and an average age of 27.1 years. The study provides clear indications that the FLOSS community is in majority young, on average in the mid-twenties [18]. Also, this study has revealed the age at which participants join the communities. Participants join FLOSS communities at ages ranging between 10 and 55 years [18] with 7% of this proportion starting below the age of 16, 33% between 16 and 20 years old, an additional third of this population between 21 and 25, 25% older than 26 for an overall average of 22.9 years old starting age [1, 18] as indicated in Table 1.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14-23 years</td>
<td>13.1</td>
<td>29.9</td>
<td>39.4</td>
<td>43.8</td>
<td>46.1</td>
<td>52.1</td>
<td>52.5</td>
<td>40.5</td>
<td>35.0</td>
</tr>
<tr>
<td>24-30 years</td>
<td>50.2</td>
<td>48.5</td>
<td>45.9</td>
<td>42.3</td>
<td>37.9</td>
<td>35.3</td>
<td>33.1</td>
<td>43.2</td>
<td>40.2</td>
</tr>
<tr>
<td>&gt; 30 years</td>
<td>36.7</td>
<td>21.6</td>
<td>14.7</td>
<td>13.9</td>
<td>16.1</td>
<td>12.5</td>
<td>14.4</td>
<td>16.2</td>
<td>24.9</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

In terms of Education, Glott et al. [1-2] and Ghosh et al. [18] reveal that FLOSS members are people with higher education. The figures indicate that 9% of the participants included in the studies’ survey had a doctorate degree while 70% boasted a university degree. Studies also suggest that 17% of the developers in FLOSS environments are high school diploma holders [19] with the rest either still schooling or having a lower educational background [19].

Furthermore, studies also reveal the professional background of these participants to better understand them. Ghosh et al. [18] as well as Ghosh and Glott [19] reveal a professional structure for FLOSS community members as depicted in Figure 1. As expected, a glance at the structure shows a great dominance for people with IT/Computer Science background as 83% of participants are representative of the sector. One third of the surveyed population is made up of software engineers, while students make up the second largest group with 16% as trailed by programmers and IT consultants. The remainder of
the group made up of executives, marketing and product sales experts do not really have a significant impact on the professional structure of the FLOSS community [1, 19].

Furthermore, 4.8% of the participants represent not currently employed people, while 14.5% is a figure for full-time students and 80% of FLOSS population is made up of students with an IT-related job [19]. Finally, 73% of participants are full-time professionals, with 14% of membership representing people who are self-employed.

1.2.1.3 Roles and Responsibilities of FLOSS members

The bulk of reports on FLOSS members profiling such as Glott et al. [1-2], Ghosh et al. [18], Ghosh and Glott [19] and the analysis performed by Krishnamurthy [20] among others, have found that OSS members in these communities hold different roles that define their responsibilities and participation in the communities activities. Among FLOSS members, project initiators and the core development team remain at the heart of any development project in the community.

Krishnamurthy [20] performed an analysis of the top 100 FLOSS communities of Sourceforge.net [20] and the findings of his work indicate that the core development team is made up of a small number of developers while the majority of the core team, referred to as the enhanced team, perform additional tasks such as feature suggestions, testing and query handling [20].

In their work on this subject, Crowston and Howison [178] developed an onion-like structure that depicts hierarchies among FLOSS members’ roles as shown in Figure 2a. The model is structured as follows:

- The core developers make the center of the model as they represent the engine behind any development project. These developers provide most of the code and monitor the entire development cycle of projects.
• The next ring represents the layers made up of co-developers. These are people who contribute to the project through patches such as bug fixes, bug reporting or even coordinating some aspects of the project. These layers also include community and project managers.

• The following layer is composed of active users. These are members of the community who do not necessarily contribute to the project in terms of coding, but their support is made through testing and bug reporting.

• The outer layer of the model represents the passive users who do not leave any tangible trace of their participation whether through forums or mailing lists.

Figure 2. An organizational structure of a typical FLOSS community

This model suggests a progressive skills development process that can be observed in FLOSS. Participants increase their involvement in the project through a process of role meritocracy [21]. This implies that passive users could move from their state of passiveness to active users until they become part of the core team [2, 21] as described in Figure 3. All these roles represent crucial contributions required for the overall project quality. However, in FLOSS environments, it is regarded as a reward and recognition of members’ abilities and contributions to move from one state to another [21].

On studying role migration and advancement in some FLOSS projects such as Mozilla, Apache, and NetBeans projects, Jensen and Scacchi [22] note the limitations of the onion model in representing the roles hierarchies in FLOSS communities. Their findings suggest that the onion model fails to represent the possible tracks of project career advancement that can be observed through different role-sets [1-2, 22]. Hence, they proposed an integrated onion-pyramid as shown in Figure 2b to further explain role migration in FLOSS.
1.2.1.4 Collaboration and Contribution activities of FLOSS members

In order for one to study learning patterns in FLOSS activities, it is helpful to understand how these activities occur. In analyzing collaboration and contribution of FLOSS members, Cerone et al. [23] describe how collaboration and quality of FLOSS participants’ contribution can be extracted from repositories. Cerone et al. argue that FLOSS repositories such as versioning systems, mailing lists and reporting systems can be examined to identify the identities of members involved in a particular communication channel, the topics of the interaction, the volume of data and information exchanged during the interaction [23]. These repositories can also provide the degree of participation in terms of number of code commits, bug fixing, email postings as well as reports and produced documentation [23].

In order to explain how collaboration and individuals contribution occurs, Cerone et al. [23] propose an Individual-team interplay model adapted to FLOSS where interaction consists of posting related activities. In this model, the product of the task is delivered through commits made by either individuals, or team leader’s approval or release decision as seen in Figure 4.
Figure 4. Individual Team interplay for FLOSS

The model in Figure 4 helps define metrics for collaboration effectiveness in FLOSS communities. Due to the evolving nature of OSS environments, Cerone et al. [23] propose to take measures for collaboration level consecutively at intervals of duration $\Delta t$. In order to get a sense of collaboration contribution, an average of the accumulated measures need to be calculated over the entire community lifetime. In Figure 4, the following numbers are considered:

- $L(t)$ of learning activities,
- $C(t)$ of contributions,
- $D(t)$ of team decisions.

The numbers occur during the period $[t, t + \Delta t]$ for a given time $t$ accounting for three stages of participation as described later in subsequent sections.

Furthermore, Cerone [11] indicates that participation in a FLOSS project evolves through three stages: Understanding, Practice and Developing. In the first stage, the purpose of interaction and communication is solely to help capture, describe and understand contents with no production activity; in the second stage, communication is used to gradually propose new contents and activities aim to defend the contents and provide feedback to other posted contents with production only at a trial and error process level; in the last stage, participants are able to get involved in development (software).

In order to study patterns of contribution from OSS members, one needs to trace participants’ contribution from all repositories containing information about projects. FLOSS developers are characterized as mobile and tend to make use of different identities in their contributions to projects [24,177]. This poses a challenge for analysts that intend to track the location (where) and levels of participation (contribution) of such developers across interrelated repositories. A number of studies have investigated this issue and provide different mechanisms to study patterns of contribution of developers from multi-repositories perspective [24-26,177].
Sowe and Cerone [24] study the integration of data from multiple FLOSS repositories and present a methodology to study patterns of contribution of 502 developers in both SVN and mailing lists in 20 GNOME projects [24]. This study notes that FLOSS developers move from one project to another while committing pieces of code, reporting and fixing bugs and taking part in discussions occurring in different IRC channels, mailing lists and forums [24]. Hence, in order to track a developers contribution in FLOSS community, their participation needs to be traced in every repository to reconstruct the identities of people involved in interactions, the subject or content of the discussions, how much developers contribute in terms of code commits or email postings etc. [24]. Moreover, the exact balance between various kinds of contributions (e.g., coding versus posting) can be discovered using Bayesian Networks [177].

Most studies conducted make use of data from CVS or mailing lists. Source Code Management (SCM) systems like CVS or SVN are used mainly for coding activities coordination purpose [24],177]. They also help manage software builds and releases. Mailing lists, as mentioned in Sowe and Cerone’s work, serve as the principal channels of communication. On these lists important details related to projects are discussed. Such details range from configuration, future software requests and tasks distribution, to meetings schedules etc. [24,177]. Thus, understanding developers’ contribution requires looking across the different tools used in coordinating projects in the community.

To exemplify this suggested approach, Sowe and Cerone [24] made use of repositories from the FLOSSMetrics[185] project and proposed a methodology that can be applied when tracking users’ participation in both SVN and mailing lists. This means that their methodology helps identify developers who make contributions both by committing code to SVN and posting messages to mailing lists. More importantly, it helps establish that developers committing codes are the same ones posting messages for the same projects. Figure 5 summarizes the methodology.

![Figure 5. Methodology to Identify developers from multiple repositories](image)

Furthermore, this methodology ensures that developers with changing identities are only counted once. The depiction of the methodology in Figure 5 shows the FLOSSMetrics database from which data from SVN and mailing lists were extracted. The figure further shows MYSQL database tables and fields that
store details of committers as well as their corresponding contribution for both SVN and mailing lists. The links between the tables as indicated by the arrows shows the path taken to locate a developer and counting his contributions to both SVN and mailing lists [24,177]. A combination of tools can be used for data extraction from FLOSS projects such as GNOME, Apache or Sourceforge [24,177]. Amor et al. [53] and Sowe et al. [54] make use of a tool called CVSanalY2 for data extraction. The tool retrieves SCM data and represents the attributes of committers in tabular format to help store the details into various tables. Sowe et al. [54] makes use of an additional tool called MLStats to extract data from mailing lists archives.

For each of the 20 projects used for analysis, committers SVN identifications (commit ID) and the total number of commits each committer made is extracted [24,177]. For the mailing list data, for each project, data was extracted from two FLOSSMetrics database tables: Two fields (type of recipient and email address) from the “messages people” table. The type of recipient field has the format “From”, “To”, and “Cc”. The “From” email header is used to identify lists posters and counting their contribution to mailing lists and three fields (email address, name, and username) from the “people” table.

Sowe and Cerone [24] suggest that due to the individuality of projects, there is a tremendous variation in developers’ contribution to coding and posting email messages [24]. However, despite these variations, the study supports that developers equally contribute to both SVN and mailing lists. Hence, it can be noted that FLOSS developers, apart from providing coding, are also involved in knowledge dissemination in the mailing lists [24,177].

While understanding patterns of contribution in FLOSS provides more lights on the activities in this environment and helps study and assess the quality of FLOSS products, we think in the context of the current work that it is an important step towards understanding how learning occurs and knowledge is disseminated between users in online development communities. In the next section, we review some works done pertaining to learning in FLOSS environments.

1.2.2 FLOSS Communities as Learning Environments

As indicated in the introduction and some subsections of this document, there has been a considerable amount of work done in exploring learning in FLOSS communities [1-11]. In most of these works and as highlighted by Cerone et al. [23], most of these findings are based on content data generated through surveys and questionnaires or through reports from observers who have been part of the community for a defined period of time. Our work will attempt to fulfill the need to provide more objective observations through empirical analysis of data from the FLOSS repositories. However, we think that it is critical to review some of these works in order to pave the way for empirical analysis.

1.2.2.1 Learning and Activity Patterns in FLOSS

Cerone [11] studies the interplay between learning and activity patterns during members’ participation and collaboration in Free/Libre Open Source Software (FLOSS) communities. His study notes the manner in which participants’ activities facilitate a learning process that possibly occurs in FLOSS members through their participation in the communities’ cycle of interactions and collaboration. The value of such analysis is, in the context of our work, to provide useful insights on how to identify and analyze learning patterns in OSS environments at both the individual and community levels.

FLOSS communities are made up of heterogeneous groups of individuals with different backgrounds, who organize themselves in an environment where they each play specific roles, with a set of
responsibilities. These participants develop different levels of knowledge that determine their participation and contribution in the community and are driven by a large variety of intrinsic and extrinsic motivations [11]. The activities in these forums are motivated by the need to bolster reputation. The reputation that members build drives the emergence of driving personalities and forms of leadership [11] in the self-organization structure created in the community.

With the extensive peer-review and collaborative discussions, OSS projects can be considered as learning and development environments where knowledge is disseminated through constant discussions and put in practice through practical contributions to software development, code revision and testing [6].

To study the link between learning and activities, Cerone [11] review typical FLOSS contributors’ roles in relation to basic activities. These roles include:

- **Observer**: This is the passive user as previously described. Although passive users do not leave any trace of their presence in the community, they do perform a number of activities such as reading product reports, related documentation and user manuals. One would also assume that observers browse through the data in repositories, looking at conversations and flow of messages in discussion forums or mailing lists, with no active participation;
- **Supporting user**: This role refers to the active user. This is the user that downloads the software releases from repositories and contributes via reporting bugs, providing feedback, helping new users, recommending the project to others, requesting new features, but does not produce artifacts;
- **Developer**: This role describes the co-developers and core team members who are at the center of any development project. They actively ensure the life of the product through coding, updating software and managing related activities;
- **Tester**: This role resembles the supporting user but is distinct in that it focuses mostly on actively testing pieces of code, reporting and possibly fixing bugs;
- **Translator**: This user helps to widen the use and access to software and related documentation to different users from different countries and languages;

Based on the above roles and responsibilities of FLOSS contributors, four basic activities can be identified [11]:

- **observe** reports, documentation, tool manuals, data, posts, code;
- **use** code, tools;
- **post** questions, requests, advises, critics;
- **commit** software, documentation, artwork, bug reports, fixed code, and translations.

The observer can formally be understood as performing two basic activities from the above four: **observe** and **use**. As Cerone [11] notes, this user is learning, as this is usually the main goal during the first stage of the participation in projects. The active or supporting user in providing feedback, recommending software, reporting bugs performs another basic activity called **post**. One can notice the active user’s comments in discussion forums, emails on the mailing lists etc. The remaining roles are both active and productive in the sense that not only do those users actively participate in discussion forums, mailing lists, wikis etc., they also produce artifacts as a result of their contribution. Developers contribute software code and documentation while testers provide bug reports and fixed code and translators provide interpreted documentation. These are instances of the last basic activity called **commit**. This is defined as the process of contributing artifacts to the FLOSS project in the forms of pieces of code, or software components. This activity can be accomplished directly if the committer has the credentials to do so or it can be done through a process spearheaded by the project leader or initiator [11].
Cerone argues that the way in which these activities are combined in order to define a participant activities pattern is bound to a number of factors including intrinsic and extrinsic motivations, maturity levels, technical and social skills. This yields a large heterogeneity of activity patterns at both individual level and community level [11]. Interactions and message exchange in discussion forums are triggered by post activities and this interaction process has two components [11]:

learning sub-process where knowledge exchange between an individual entity and the rest of the community increases knowledge acquisition at both the individual level and the team or community level; contribution sub-process in which a contribution in the form of commit is a direct result of communication and message exchange.

Hence through this interaction process, one can identify and note the presence of a collaborative learning process, that consists of collaborative peer-review and criticism through which a wealth of knowledge is built. Therefore, “contribution, which is based on commit, is the result of communication, which is based on post, individual learning, which is based on observe and use, and collaborative learning, which is based on post” [11].

Furthermore, “the interaction process is cyclic on its two components, in the sense that both the knowledge that results from the learning sub-process and the artifacts that result from the contribution sub-process feed a new iteration of the interaction process [11].

Having identified the basic activities in relation with the participants’ roles and responsibilities, as described by Cerone [11], we understand that OSS communities provide a platform in which resources and actors enable the definition of learning processes. The social context defined in these virtual environments supports acquisition of knowledge that eventually translates into participants’ volume of contribution. For example, the ability to develop source code is enriched through observing, studying, reviewing and modifying existing code from extensive discussions with other members of the community. Through discussion and documentation, the ability to interact and make use of existing tools is enabled.

FLOSS contributor’s learning process occurs through four phases through which knowledge evolves [27]. These phases include socialization, externalization, combination and internalization [27]. As figure 6 illustrates, OSS community members exchange knowledge while socializing and make their tacit knowledge explicit through externalization. This explicit knowledge leads to the formation and organization of abstract knowledge through combination. Actors internalize the explicit knowledge and generate “new” tacit knowledge by combining it with their own observations and knowledge. Actors’ tacit knowledge becomes explicit through intensive participation.
Process Models for Learning patterns in FLOSS repositories

Chapter 1: Introduction

Knowledge exchange in FLOSS communities occurs in synergy. If an actor shares information about an effective method of implementing some aspect of the software for example, this process of putting ideas into words will help the actor to shape and improve these ideas [27]. When participants engage in discussions with the community, they share ideas and learn from each other. This knowledge exchange is logged into FLOSS repositories and can serve as learning ground to future users.

Cerone [11] links the four phases in Figure 6 to the basic activities as shown in Figure 7 and argue that socialization is enabled by post; post, commit enable externalization while combination of community explicit knowledge and its organization as abstract knowledge is enabled by observe and use; finally, internalization leads to generation of new tacit knowledge.

Figure 6. Making tacit knowledge explicit in FLOSS communities
To conclude this section about learning and activity patterns, we mention the categorization of learning stages in addition to the different phases participants go through during knowledge exchange. Cerone [11] identifies and distinguishes three stages that characterize the evolution of activity patterns and maturation of an OSS community participant:

— **Understanding**: This is the initial stage of the learning process during which participants get involved in the projects by reviewing, communicating with the purpose of understanding contents without producing any tangible contribution. This is a critical stage as the participant accesses project repositories and exchanges emails and posts messages to get acquainted with the contents of the repositories.

— **Practicing**: During this stage, the participant evolves from understanding to providing new contents in discussions forums, defending these contents and criticizing the existing materials.

— **Developing**: at this stage, the participant is able to produce and review contribution from peers in terms of coding and software artifacts.

Figure 8 describes how these learning stages can be linked to basic contributor’s activities. The curve depicts the contributor’s evolution through the three stages of learning process [11]. The figure also indicates for each learning stage, the corresponding activities that the contributor is able to undertake during that stage. Cerone notes that the *maturation* process evolves in a gradual and continuous way [11]. Figure 9 on the other hand, depicts how specific activities (referred to as basic activity contents) performed by contributors affect their growth through learning stages.

**Figure 7. Learning process of individual actors in OSS communities**
Figure 8. Learning Stages and contributor’s activities in OSS communities

Figure 9. Learning Stages and Activity contents
1.2.2.2 FLOSS as E-learning Tools

As findings that support the occurrence of learning within FLOSS environments increase, practitioners in tertiary education have attempted to incorporate participation in FLOSS projects as a requirement for some Software Engineering courses [6]. A number of pilot studies have been conducted so as to evaluate the effectiveness of such an approach in traditional settings of learning [6, 28, and 30].

FLOSS environments currently present an alternative approach that enables students to work on real-world problems for a more effective learning of software engineering as suggested by the joint IEEE/ACM CS undergraduate curriculum guidelines [29]. In this section we report on some results obtained from two pilot studies that were conducted for teaching Software Engineering at undergraduate and postgraduate levels [30]. We also look at findings from a similar experiment on a large scale for a longer period of time as presented in [28].

The first pilot study was conducted by Sowe and Stamelos [30] and the objective was to assess whether FLOSS projects could be used as part of teaching Software Engineering courses as part of the curriculum in a formal learning. This study builds on an initial investigation [31] aimed at teaching software testing, in order to help develop a methodology on how to assess student participation [30]. The study was conducted at Aristotle University in Greece where 15 students from a total of 150 enrolled students to “Introduction to Software Engineering” course participated in the study and 13 of whom completed. It consisted of three phases. The first phase was about introducing students to FLOSS environments through lectures. During this period, students had access to projects in FLOSS, they browsed through the projects and made a choice on which one to join; the second phase required students to participate in the chosen project in order to undertake a number of activities including finding and reporting bugs and possibly fixing them while the last phase was about students evaluation by their lecturers.

At the completion of this study, two surveys were conducted among the students that took part in the pilot study, and the results indicated that most of them expressed the desire to prolong their participation in these FLOSS projects even after their graduation. Consequently, students continued to report on their activities to their lecturers after the conclusion of the pilot study [6].

The second study was conducted at postgraduate level by Jaccheri and Østerlie [33]. Their approach consisted of involving master’s students to investigate and study the documentation and literature in OSS development and come up with possible research questions that can be solely addressed by participation in a FLOSS project. Students would then select a project related to the assignment and formulated research questions, then join the project as developers and also act as researchers in addressing the research questions. The results of this study are presented for a master student required to take part in a commercially controlled OSS project, the NetBeans open source project, in order to study the various benefits that the project provides to firms [33]. The student was requested to determine “how the use of Software Engineering techniques, such as explicit planning, ownership, inspection and testing, affects the OSS project” [6].

Papadopoulos et al. [28] present findings after four years of using FLOSS projects as tools to enhance software engineering teaching in formal learning. The experiments were conducted at the Aristotle University in Greece on 408 junior students in Informatics [30]. The respondents assumed different roles representing some aspects of software engineering such as requirements engineers, testers, developers, and designers/analysts. This study involved students enrolled for two courses namely “Introduction in Software Engineering” (ISE) and “Object-Oriented Analysis” (OOA) offered as part of the curriculum in a 4-year degree program. With about 150 enrolling every semester, the number of students participating in the experiment increased every semester over the duration of the study. Just like in the first pilot study, students were introduced to some background information on FLOSS and allowed time to get acquainted to the concept before they join a project of their choice to work on the assignments. In the first course, the assignments require students to adopt the roles of requirements engineer, tester, or developer, while they
can choose to be either designer or analyst for the second course. This diversity of roles is intended to allow students the opportunity to experience FLOSS environments from different perspectives. This experience is vital for the entire class as these students regularly report to their classes on their activities. Furthermore, students were not restricted to specific roles as they were allowed to take up different responsibilities over the course of their involvement in FLOSS; testers could become developers and assist in fixing and providing code and so forth. However they were not allowed to play the same roles in the same projects.

A learning environment was used to coordinate students’ activities during the time they undertook to perform their tasks and even after this period when they helped and shared their experiences. This environment also served as integrated communication platform among students with supports for FAQ section, roles sub-forums, past experiences, managerial issues section supported by instructors [28]. The findings of this experiment indicate that students were able to accommodate to the new environment and successfully carry out their activities although working on real problems in a real world context presented both challenges and motivations for them [28]. The results over the course of this investigation were positive as students could perform their tasks and undertake FLOSS activities. Moreover, most students indicated their desire to continue being part of the communities they joined after the completion of the assignments. They also indicated that participation in FLOSS environments fosters skills development and knowledge acquisition.

In these studies, the freedom given to students for the selection of the project is critical in ensuring that they can finally complete their task. FLOSS environments are heterogeneous communities of “volunteers” and this constitutes an important aspect that needs to be preserved while carrying out pilot studies. Participants are endowed with different levels of skills and competency and this drives the decision for project choice.

Furthermore, as the number of such experiments increases, Meiszner et al. [36] also propose 3 approaches to incorporating FLOSS projects in a traditional setting of learning:

— The first approach is called the ‘inside approach’. Here, FLOSS environments principles are copied and applied to learning in higher institutions settings. The practice suggests creation of virtual learning environments by these institutions which can be open to different stakeholders (students, lecturers and other users) to exchange and participate in their activities under a controlled environment.

— The second approach is referred to as the ‘outside approach’. With this method, students are requested to join well-established and existing environments and perform a predefined number of tasks on the basis of which they are evaluated. In this context, students are introduced to the fundamentals of FLOSS environments to equip them with theoretical background before being assigned tasks.

— The last approach is a blend of the first two approaches. This appears to be the most complex to implement because it might require a drastic overhaul of higher educational practices [28].

Another important aspect in understanding the context in which learning opportunities occurs in FLOSS environments pertains to FLOSS participants. Some studies have been conducted in order to investigate the extent to which FLOSS community members are aware of this occurrence. We report on some findings on this regard in the next subsection.

1.2.2.3 Learning Awareness

In light of findings presented in previous sections, it can be argued that active participation in FLOSS through messages exchange, code review or documentation writing can help foster learning. Fernandes et al.[34] provide more insights on this argument. In this study, a questionnaire was designed to reflect on contributors’ experience in FLOSS projects participation. More precisely, the aim was to analyze from
FLOSS contributors’ perspective if they recognized FLOSS environments as learning communities in light of their activities [34]. It was conducted within the context of the Common-based Peer Production (CBPP) model. This model defines collaboration efforts from a large group of people that work incrementally towards a problem while sharing information [34].

Dillon and Bacon [35] identify 4 key characteristics of such a model that represents FLOSS environments. The first trait is motivation in these environments. Dillon and Bacon [35] reveal that money is not the main reason why participants join a project. The second characteristic is the existence of chunking in these environments. This implies that individual groups incrementally and asynchronously work on tasks. The third trait of FLOSS is multi-disciplinary. There is a diversity of skill sets and expertise from contributors that work on different aspects of the project. Finally, FLOSS environments provide opportunities for integration of different products [35].

The questionnaire was designed to address questions about participants’ ability to interact with others, the recollection of their own achievements, their commitment level and most importantly their level of awareness on the existence of learning processes within the community [34]. The study survey was conducted on a limited number of subjects. The results are based on responses from 3 contributors. The first participant is a PhD male student and has been member on a number of projects including Perl, Parrot and Debian; the second respondent is a full-time male software developer with a degree in Mathematics and computer science who has worked on projects such as gwibber, bigodejs, plone and django; the last participant is a male who holds a PhD in computer sciences, and currently works as a NLP researcher and has contributed on a number of FLOSS projects including Perl Dancer, Perl Lingua::Jspell, Perl Lingua:: NATools and Perl Lingua::FreeLing3.

The results reveal that these 3 contributors unanimously agree that FLOSS environments provide a learning platform. Their responses also provide further hints on opportunities to foster their skills they could not afford during their formal university studies. FLOSS projects provide opportunities to improve their skills on areas such as software usability. More importantly, we can highlight their view of knowledge sharing and skill development that take place in FLOSS environments. This view entails that through collaborative participation, knowledge is naturally and easily brokered within FLOSS communities.

As the authors note, this is a preliminary investigation that relies on weak results because it was conducted through an online questionnaire, answers may not have been entirely reliable. Hence, this requires further validation. We hope to contribute towards these observations with empirical evidence based on data from FLOSS repositories.

1.2.2.4 Learning according to FLOSScom

To close the review on learning in FLOSS environments, we highlight some of the main findings from FLOSScom on their analysis of informal learning in these communities [2]. The results of this study are based on a survey where 350 participants were asked to fill a questionnaire on skills acquired in FLOSS. According to this study, there are 12 levels (cohorts) of professional expertise that testified to have learned something in FLOSS. There is an indication of a learning curve among the young cohorts as their learning process is gradual and expanding through skills improvements for different categories of skill set. This learning curve progressively starts with social skill when they express their opinions and interact [11, 27].

Results also indicate that the semi-experienced participants significantly increase their skills like the first cohort but with a special mention in programming skills [2]. The cohort of experienced participants demonstrates similar skills improvements but they also excel in managerial and legal skills. Unlike the
young cohort that shows signs of learning curve, the middle-aged cohort show sign of skills acquisition with a focus on specific skill sets.

In short, the results of this study indicate that participants in FLOSS communities acquire skills differently because of their professional background and experience. More importantly, apparently young community members with no professional experience agree that FLOSS communities provide an adequate learning environment that provides for an informal but very comprehensive curriculum capable of making them full-fledged FLOSS programmers, coordinators and activists [2].

1.2.3 Process Mining Techniques for Knowledge discovery

Process Mining is used as a method of reconstructing processes as executed from Event Logs [38]. These logs are generated from process-aware information systems such as Enterprise Resource Planning (ERP), Workflow Management (WFM), Customer Relationship Management (CRM), Supply Chain Management (SCM), and Product Data Management (PDM) [14]. The logs contain records of events such as activities being executed or messages being exchanged on which Process Mining techniques can be applied in order to discover, analyze, diagnose and improve processes, organizational, social and data structures [37]. This can also be understood as the automated discovery of processes from Event Logs resulting in the generation of a process model (e.g., a Petri net) that describes the causal dependencies between activities [14].

More specifically, Van der Aalst et al. [14] describe the goal of Process Mining to be the extraction of information on the process from Event Logs using a family of a-posteriori analysis techniques. These techniques enable the identification of sequentially recorded events where each event refers to an activity and is related to a particular case (i.e., a process instance). They can also help identify the performer or originator of the event (i.e., the person/resource executing or initiating the activity), the timestamp of the event, or data elements recorded with the event. Such information is critical in our endeavor, as we attempt to study the generation and originators of learning patterns from Event Logs we built from FLOSS repositories.

Process Mining comprises three different techniques: discovery, conformance and extension [14]. In Discovery, there is no a-priori model, neither an existing model; the “discovered” model instead stems from an Event Log. An example is shown in Figure 10: a process model generated using the $\alpha$-algorithm (described later). Conformance refers to an a-priori model that is used to verify if the events recorded in the log conform to it; this is used to detect deviations, locate and explain them in order to take appropriate actions. The last technique is Extension where there is an a-priori model that is extended or enriched with new aspects; for example the extension of a process model with performance data [14, 37, and 39].

Current Process Mining techniques evolved from the work done by Weijters and Van der Aalst [38] where the purpose was to generate a workflow design from recorded information on workflow processes as they take place. This is accomplished based on an assumption that in Event Logs, each event refers to a task (a well-defined step in the workflow), each event refers to a case (a workflow instance), and these events are recorded in a certain order. Combining techniques from machine learning and Workflow nets, the authors construct Petri nets that provide a graphical but formal language for modeling concurrency [38]. Figure 10 depicts an example of a workflow process modeled as a Petri net.
In Workflows, as shown in Figure 10, cases or workflow instances represent activities such as an insurance claim, a tax declaration, or a request for information [38]. These cases are handled through executing tasks in a specific order. They enable this execution by specifying which tasks need to be executed and in which order, as exemplified in the Petri net in Figure 10. A Petri net helps in routing the execution of cases. Hence, transitions model tasks while places and arcs model causal dependencies. In Figure 10, the transitions T1, T2, ..., T13 represent tasks, The places Sb, P1, ..., P10, Se represent the causal dependencies. Places can also be understood as representing pre or post-conditions for tasks. An AND-split corresponds to a transition with two or more output places (from T2 to P2 and P3), and an AND-join corresponds to a transition with two or more input places (from P8 and P9 to T11). OR-splits/OR-joins correspond to places with multiple outgoing/ingoing arcs (from P5 to T6 and T7, and from T7 and T10 to P8). At any time a place contains zero or more tokens, drawn as black dots.

We consider a workflow log as a set of event sequences where each event sequence is simply a sequence of task identifiers [38], formally expressed as WL \( \subseteq T^* \) where WL is a workflow log and T is the set of tasks. An example event sequence of the Petri net of Figure 10 is: T1, T2, T4, T3, T5, T9, T6, T3, T5, T10, T8, T11, T12, T2, T4, T7, T3, T5, T8, T11, T13. Hence, one can conclude that given a workflow log WL, Process Mining enables the discovery of a WF-net that (i) potentially generates all event sequence appearing in WL, (ii) generates as few event sequences of \( T^* \setminus WL \) as possible, (iii) captures concurrent behavior, and (iv) is as simple and compact as possible. The Process Mining technique used to generate the Petri net consists of three distinct steps: Step (i) the construction of a dependency/frequency table (D/F-table), Step (ii) the induction of a D/F-graph out of a D/F-table, and Step (iii) the reconstruction of the WF-net out of the D/F-table and the D/F graph [38].

Significant advances in Process Mining have been done with regards to algorithm implementation and tool development [14, 37, 39-42]. Due to space constraints, we cannot report on all of these as well as the steps of Process Mining in generating such a workflow model as depicted in Figure 10. We find the work by Van der Aalst et al. [39] to be an important resource in this regard. Nevertheless, we note that the discovery process in Process Mining is carried out using the \( \alpha \)-algorithm, which implements consecutive reads of a given Event Log, identifies and gets the set of tasks, infers the ordering relations, builds the net based on inferred relations and outputs the net. Another algorithm called the Multi-phase approach can also be used for Process Mining as detailed in [47]. However, we believe that the preliminaries of Process Mining can be easily understood starting with the \( \alpha \)-algorithm of which formalization is given below.

Let W be a workflow log over T. \( \alpha(W) \) is defined as follows.
1. \( T_W = \{ t \in T \mid \exists \sigma \in W \ t = \sigma \} \),
2. \( T_I = \{ t \in T \mid \exists \sigma \in W \ t = \text{first}(\sigma) \} \),
3. \( T_O = \{ t \in T \mid \exists \sigma \in W \ t = \text{last}(\sigma) \} \),
4. \( X_W = \{ (A,B) \mid A \subseteq T_W \land B \subseteq T_W \land \forall a \in A \ \forall b \in B \ a \rightarrow_W b \land \forall a_1.a_2 \in A \ a_1 \#W a_2 \land \forall b_1.b_2 \in B \ b_1 \#W b_2 \} \),
5. \( Y_W = \{ (A,B) \mid \forall (A',B') \in X_A \subseteq A' \land B' \subseteq B' \Rightarrow (A,B) = (A',B') \} \),
6. \( P_W = \{ p(A,B) \mid (A,B) \in Y_W \} \cup \{ i_W, o_W \} \),
7. \( F_W = \{ (a,p(A,B)) \mid (A,B) \in Y_W \land a \in A \} \cup \{ (p(A,B),b) \mid (A,B) \in Y_W \land b \in B \} \cup \{ (i_W,t) \mid t \in T_I \} \cup \{ (t,o_W) \mid t \in T_O \} \), and
8. \( \alpha(W) = (P_W, T_W, F_W) \).

Understanding the basic notations in the algorithm requires a review of some fundamentals and preliminaries on Petri nets and Workflow nets [39, 44]. First, we review some classical definitions of \( P/T \).

**Definition 1 [39]:** (P/T-nets) is a Place/Transition net, or simply a P/T-net, is a tuple \((P, T, F)\) where:
1. \( P \) is a finite set of places,
2. \( T \) is a finite set of transitions such that \( P \cap T = \emptyset \), and
3. \( F \subseteq (P \times T) \cup (T \times P) \) is a set of directed arcs, called the flow relation.

**Definition 2 [44]:** A net is \( PN = (P, T, F, W, M_0) \) where:
1. \( P \) is a finite set of places,
2. \( T \) is a finite set of transitions such that \( P \cap T = \emptyset \), and
3. \( F \subseteq (P \times T) \cup (T \times P) \) is a set of directed arcs,
4. \( W \) is a weight function of arcs, (default = 1)
5. \( M_0 : P \rightarrow \{0, 1, 2, \ldots\} \) is the initial marking where \( P \cap T = \emptyset \) and \( P \cup T \), \( \emptyset \).
6. \( k = P \rightarrow \{1, 2, 3, \ldots\} \cup \{\infty\} \) = partial capacity restriction (default = \( \infty \)).

**Definition 3 [44]:** Let \( X = P \cup T \) and \( N = (P, T, F, W, M_0) \) be a PN, then:
1. \( \bullet x = \{ y \in X \mid (y, x) \in F \} \) is the pre-set (input set) of \( x \),
2. \( x^\bullet = \{ y \in X \mid (x, y) \in F \} \) is the post-set (output set) of \( x \),
3. \( \text{nbh}[x] = \bullet x \cup x^\bullet \) is called neighborhood of \( x \),
4. If \( Y \subseteq X \) then \( \bullet Y = U \bullet x \) and \( Y^\bullet = U x^\bullet \).

**Definition 4 [44]:** Let \( N = (P, T, F, W, M_0) \) be a PN then PN:
1. is P-simple \( \forall x, y \in P, (\bullet x = \bullet y \land x^\bullet = y^\bullet \Rightarrow x = y) \)
2. is T-simple \( \forall s, t \in T, (\bullet s = \bullet t \land s^\bullet = t^\bullet \Rightarrow s = t) \)
3. has no isolated places \( \forall x \in X, \text{nbh}(x) \), \( \emptyset \)

**Definition 5 [44]:** A PN is:
1. pure \( \forall x \in X, [\bullet x \cap x^\bullet = \emptyset] \),
2. simple \( \forall x, y \in X, [(\bullet x = \bullet y \land x^\bullet = y^\bullet ) \Rightarrow x = y] \)
Additional notions crucial for understanding of Petri nets include the firing rule, reachable markings, firing sequence, connectedness, boundedness, safeness, dead transitions and liveness [39].

When a Petri net is constructed to model a workflow, it is called a Workflow net (WF-net) [39]. Let $N = (P, T, F)$ be a P/T-net and $t^*$ be a fresh identifier not in $P \cup T$. $N$ is a workflow net (WF-net) iff:
1. Object creation: $P$ contains an input place $i$ such that $i^* = \emptyset$,
2. Object completion: $P$ contains an output place $o$ such that $o^* = \emptyset$,
3. Connectedness: $N^* = (P, T \cup \{t^*\}, F \cup \{(o,t^*),(t, i^*)\})$ is strongly connected.

Furthermore, the concept of workflow mining, whose purpose is to produce a workflow net on the basis of a workflow log such as the one given in Figure 11, can be understood by analyzing workflow logs for Workflow nets generation as briefly described below. Table 2 shows an example of a workflow log.

<table>
<thead>
<tr>
<th>Table 2. A workflow log</th>
</tr>
</thead>
<tbody>
<tr>
<td>case identifier</td>
</tr>
<tr>
<td>case 1</td>
</tr>
<tr>
<td>case 2</td>
</tr>
<tr>
<td>case 3</td>
</tr>
<tr>
<td>case 3</td>
</tr>
<tr>
<td>case 1</td>
</tr>
<tr>
<td>case 1</td>
</tr>
<tr>
<td>case 2</td>
</tr>
<tr>
<td>case 4</td>
</tr>
<tr>
<td>case 2</td>
</tr>
<tr>
<td>case 2</td>
</tr>
<tr>
<td>case 5</td>
</tr>
<tr>
<td>case 4</td>
</tr>
<tr>
<td>case 1</td>
</tr>
<tr>
<td>case 3</td>
</tr>
<tr>
<td>case 3</td>
</tr>
<tr>
<td>case 4</td>
</tr>
<tr>
<td>case 5</td>
</tr>
<tr>
<td>case 5</td>
</tr>
<tr>
<td>case 4</td>
</tr>
</tbody>
</table>

From the log in Table 2, we can define a Workflow trace as follows: Let $T$ be a set of tasks. $\sigma \in T^*$ is a workflow trace and $W \in P(T^*)$ is a workflow log.

The workflow trace of case 1 in Table 2 is ABCD. The workflow log corresponding to Table 2 is \{ABCD, ACBD, AED\} because workflow traces ABCD and ACBD appear twice in case 1 and case 3 as well as case 2 and case 4 respectively.
With such information available, the sequence of execution of the α-algorithm is as follows [45]: the log traces are examined and in the first step, the algorithm creates (Step 1) the set of transitions ($T_w$) in the workflow, (Step 2) the set of output transitions ($T_i$) of the source place, and (Step 3) the set of the input transitions ($T_0$) of the sink place, the (Steps 4 and 5, respectively), the α-algorithm creates sets ($X_w$ and $Y_w$, respectively) used to define the places of the mined workflow net. In Step 4, the algorithm discovers which transitions are causally related. Thus, for each tuple $(A, B)$ in $X_w$, each transition in set $A$ causally relates to all transitions in set $B$, and no transitions within $A$ (or $B$) follow each other in some firing sequence. Note that the OR-split/join requires the fusion of places. In Step 5, the α-algorithm refines set $X_W$ by taking only the largest elements with respect to set inclusion. In fact, Step 5 establishes the exact amount of places the mined net has (excluding the source place $i_W$ and the sink place $o_W$). The places are created in Step 6 and connected to their respective input/output transitions in Step 7. The mined workflow net is returned in Step 8 [45].

From a workflow log, four important relations are derived upon which the algorithm is based. These are $>_w$, $\rightarrow_w$, $#_w$, and $||_w$ [45]. In order to construct a model such as the one in Figure 10 on the basis of a workflow log, the latter has to be analyzed for causal dependencies [39]. For this purpose, the Log-based ordering relations notation is introduced:

Let $W$ be a workflow log over $T$, i.e., $W \in P(T^*)$. Let $a, b \in T$:

- $a >_w b$ if and only if there is a trace $\sigma = t_1t_2...t_n$ and $i \in \{1,...,n-2\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$,
- $a \rightarrow_w b$ if and only if $a >_w b$ and not $b >_w a$,
- $a #_w b$ if and only if not $a >_w b$ and not $b >_w a$,
- $a ||_w b$ if and only if $a >_w b$ and $b >_w a$.

Considering the workflow log $W = \{ABCD, ACBD, AED\}$, relation $>_w$ describes which tasks appeared in sequence (one directly following the other). Clearly, $A >_w B$, $A >_w C$, $A >_w E$, $B >_w C$, $B >_w D$, $C >_w B$, $C >_w D$, and $E >_w D$. Relation $\rightarrow_w$ can be computed from $>_w$ and is referred to as the (direct) causal relation derived from workflow log $W$. In our example we have: $A \rightarrow_w B$, $A \rightarrow_w C$, $A \rightarrow_w E$, $B \rightarrow_w D$, $C \rightarrow_w D$, and $E \rightarrow_w D$. Note that $B \rightarrow_w C$ because $C >_w B$. Relation $||_w$ suggests potential parallelism.

More details about the algorithm, its potentials and limitations as well as the current status of its implementation are described further in the work by Van der Aalst and others [14, 39, 43 and 45].

### 1.3 Related Work

Process Mining as a method of workflow log analysis has been widely used in different sectors due to the prominence of the ProM support tool [40]. In light of our research, there are some studies that report on findings about the use of Process Mining in software development [13, 15 and 46].

Due to the growing demands for increased functionalities and integration complexities within different kinds of software, software engineering poses real challenges. A number of Document management Systems such as Software Configuration Management (SCM) systems (i.e. CVS, subversion) and Product Data Management (PDM) systems are used by software engineers in order to manage the amount of data, documents and files pertaining to software development [13]. Despite the existence of such mechanisms, the processes are still less structured. These systems contain not only flat files but also critical information about individuals who create access and update documents as well as the timestamp associated to the execution of these tasks [13]. Hence, mining these repositories by means of Process Mining techniques can help software engineers to identify and analyze and better understand the processes involved during software development. This is called software Process Mining [13].

The application of Process Mining to software development processes occurs in different aspects including the control, information and organization aspects [13]. The control aspect analyzes the sequential execution of tasks, while the information aspect concerns with data, documents and
information related to and produced by task. The organization aspect captures the individuals/developers roles in executing tasks [13].

The ProM framework [13] encompasses a variety of algorithms that allow for an audit trail (log) of a SCM system to be mined in order to produce a process model that can be analyzed for monitoring and improving software processes. The Social network Miner as presented by Voinea and Telea [48] can help generate a social network of individuals involved in the processes. Such a model gives a perspective on the ties and relationships and the nature of exchange that takes place between these individuals. The organizational miner of ProM helps identify roles between resources by clustering resources having similar tasks, while the Activity miner addresses the problem of information related to tasks in a given log [13].

In addition to these studies on mining software repositories, Poncin et al. [15] propose a framework called FRASR (FRamework for Analyzing Software Repositories) aimed at extending the functionalities of the ProM framework through a combination of multiple repositories content. With experiments particularly carried in OSS environments, FRASR is used to mine FLOSS developers’ activities such as their code commits, comments and emails exchange between them making use of the repositories [15]. This approach proposes to match related software development events such as emails about a specific file, modifications to that file as well as related bug reports [15]. The deliverable of this approach is a process log generated by combining such events. Once the log is produced, it can be input to ProM for appropriate analysis.

Other studies have investigated aspects of FLOSS environments through the application of Social Mining to study social interactions in these platforms [2, 49]. These studies report on identification of hidden patterns of information flow between contributors. Some of these results have helped understand how FLOSS communities are organized, how interactions are conducted, and the structure and evolving nature of sub networks in these communities, the way participants organize themselves, how tasks distribution occurs, how participants share knowledge through mailing lists and other portals of knowledge exchange etc.[2]. Social Mining also helps generate visuals of interactions in FLOSS communities and facilitates the identification of hubs or experts, top collaborators and their communication patterns. Finally, social network analysis can also be applied to mailing lists in FLOSS environments to study the exchange of messages between participants. In such studies, the number of mails contributed by each node depicts the weights associated with the tie [50].

The last results pertain to the application of Process Mining in order to construct social networks from Event Logs. The ProM tool provides functionalities that help construct and analyze social networks from workflow logs because of the existence of information about individuals involved in processes in Event Logs [13, 51-52]. Van Der Aalst et al. [52] argue that it is possible to generate and analyze social networks from Event Logs by combining techniques from workflow management and social network analysis. Their study involved an Event Log made up of 5000 cases and more than 33,000 events. And based on this information, their experiment demonstrated that that an Event Log can be used to derive relations between performers of activities, thus resulting in a social network that can be interpreted and analyzed using SAN techniques.

In spite of these interests in applying Process Mining to software repositories, limited work has been done regarding identifying and analyzing learning patterns in FLOSS environments. This gap triggers the introduction of a new alternative through our novel approach that seeks to explain how learning patterns can be traced in FLOSS repositories and the extent to which current theories can be evaluated against reality.
1.4 Proposed Approach

In line with all that has been said above, we propose an analysis of learning patterns in FLOSS environments through the application of Process Mining algorithms on data extracted from repositories (SVN, emails and discussion forums etc.). Through this analysis of FLOSS repositories, learning patterns in participants’ activities and/or messages exchange will result in constructing process models that can be validated for conformance or learning patterns discrepancy with existing approaches as depicted in Figure 11. Hence, our approach is based on a number of hypotheses:

[H1] FLOSS communities are indeed possible learning environments  
[H2] FLOSS repositories can be analyzed using Process Mining tools and techniques for learning patterns identification and representation  
[H3] New methods and supporting systems (Algorithms) for learning process identification in FLOSS can be developed and evaluated.

![Diagram](image)

**Figure 11.** A Graphical representation of our proposed problem solving approach

In our approach, we also hope to provide answers to questions raised by Cerone et al.[23]:

1. how communication and development enable a natural *learning process* ;
2. how the linkage between learning process and basic activities drives evolution of activity patterns and maturation of participants in FLOSS;
3. how activity patterns can be analyzed to identify the presence of learning patterns.

The implementation of the approach follows a non-sequential seven-step conceptual framework that will serve as our methodology as depicted in Figure 12. The steps include define learning vocabularies (Ontology), contextualize learning processes through a-priori models, generate “semantic-oriented” Event Logs, generate corresponding process models, interpret and assess the value of process discovery and finally verify the hypotheses as formulated.

![Figure 12. Methodology for Learning Patterns Identification](image-url)
A certain level of iteration is to be maintained in the execution of this framework. This means that the steps do not necessarily have to be performed or undertaken strictly in a sequential fashion; however, if needed, one can still go back and forth in order to update or adjust the deliverables as required.

In step 1, the idea is to design and agree on a common definition of concepts or key words and phrases that one can use to identify whether or not learning has taken place during an interaction between FLOSS members. These keywords will guide the rest of the process in that they will help trace a learning participants’ trails of activities in mailing lists, SVN, bug trackers and even forums in order to rate the progression of the learning process if any.

Step 2, contextualizing learning processes, provides the foundation that will guide conclusions one can make about learning patterns in FLOSS in our defined context. Looking at the existing literature about learning in FLOSS communities, we attempt to provide a graphical representation of the learning behavior as perceived to date. This representation is based on some pedagogical interpretations of how people can engage in a learning experience and facilitate skills acquisition [1-3].

In e-Learning or virtual learning communities, there are some strategies that are put in place to explain the learning exchange channels. These identify the different ways and the nature thereof through which learning can occur. These could be interpreted as instances of learning processes. These are sequences of activities that would explain how people exchange and build knowledge. In order to understand how these could be built or generated, one could consider some types of these interactions in FLOSS communities including directed and undirected learning. The second type of learning process (Directed Learning) can be described from 2 perspectives in 4 different formats. These are:

— Undirected Learning: This process can also be referred to as Peer-2-Peer or Reflective Learning. This learning is assumed to take place between any numbers of participants. In this process, any participant can be both a receiver (Learner) and a sender (Teacher). At this level, the assumption is that learning occurs between mates with a diversified expertise background who learn from each other.

— Directed Learning: This process refers to involvement of more knowledgeable participants or expert members in helping less expert members to develop their skills with some level of guidance or supervision. The occurrence of the process is twofold:
  - Pulling: This is the process where a less expert on any topic would initiate a need to learn by reaching out to the more advanced that can culminate in a supervised or guided learning process. This can also in turn occur following 4 formats as follow:
    ▪ Modeling: In this process, the guide’s (sender’s) activities and actions are systematically monitored and observed by the receiver. This can happen as the receiver aims to emulate the sender given the latter’s reputation on their FLOSS contribution. An example could be tracking the sender’s commits in SVN, their comments on mailing lists etc.;
    ▪ Coaching: As the term explains, this involves giving direct monitoring and guidance to the requester’s and observing his/her performance;
    ▪ Scaffolding: In this process, the sender analyses and determines the receiver’s level of capacity and allows him/her the opportunities to acquire knowledge accordingly. For example, supplying materials (tutorials etc.) on specific problems and a solution approach etc. based on the requester’s background.
    ▪ Fading: This process depicts involving a requester in practical execution of tasks for skills acquisition. However, as the requester’s performance matures, the sender gradually gives them autonomy to apply their skills.
  - Pushing: This is the type of directed learning that occurs when the sender takes the initiative to make available opportunities of knowledge acquisition for requesters. Just
like the pulling, this process can also be understood in 4 formats including Modeling, Coaching, Scaffolding and Fading.

These instances of learning processes occur in FLOSS communities depending on the nature of questions that trigger the exchange of knowledge and generation of skills among participating members [1-3]. Nevertheless, a given learning process can be defined as occurring through a number of phases depending on the learning stages as discussed in the previous sections. In this step of our methodology, we define and describe the resulting learning phases as per the participants’ activities through a set of Petri net-like Workflow nets that give a visual representation of these learning processes based on the current description found in the literature.

The third step of the methodology entails the preprocessing task of Process Mining. This is related to the generation of the logs that will be used to construct the process models. We refer to the Event Logs as semantic oriented because the events we will return will be based on the contextual and semantical meaning of the data we analyze. Especially when comments or messages are exchanged between FLOSS community members, it is crucial to obtain the real content of these messages for analysis and not just query their syntax. Therefore, the first two steps provide the essential foundation needed to execute this step.

The following step concerns the generation and analysis of the corresponding process models that explain how and what kind of activities govern the learning processes in FLOSS repositories. This is the first objective in terms of process discovery that is achieved through Process Mining. We produce the process maps for both the participants in the three identified phases of learning process.

The next step, interpret and assess the value of process discovery, entails explaining the extent to which the discovered process maps provide evidence of the existence of learning patterns in the Event Logs built as a result of the previous steps. The last two steps entail the evaluation of learning process execution in light of a-priori models in order to discuss discrepancies between the two sets of models. In conjunction with this task, we also conclude by showing how following this approach has provided enough evidence to verify our hypotheses.

Therefore, the purpose of this work is not simply to verify whether learning can occur in FLOSS environments but more importantly provide insights into the ways in which it does. Thus, the main objectives can be summarised as follows:

- **[obj1]** Develop and implement new techniques that can help semantically extract information related to participants’ activities during learning processes
- **[obj2]** Apply these techniques to generate semantic-based logs
- **[obj3]** Process mine these logs to produce related process models
- **[obj4]** Evaluate the models and discuss the results
1.5 Conclusion

FLOSS communities are an important phenomenon that has raised interest from different perspectives of research. Given the quality of FLOSS products, the environments appear to provide learning opportunities for participants. While this aspect has been investigated with reports produced by Glott et al. [1-2], we believe that there is a need for empirical evidence to support such findings. Cerone et al. [23] critically point out that in such reports, content data had been collected using surveys and questionnaires or through reports from observers who have been part of the community for a defined period of time. In light of this, we highlight the need to provide some more empirical evidence to support and describe the existence of learning patterns in FLOSS communities.

Therefore, in this chapter, the objectives have been to contextualize and scope our research endeavor. We motivated the need for such a research study by defining the primary objectives and scoping the work in terms of a number of hypotheses. Succinctly, we aim to study and provide supporting evidence or the lack thereof about learning in FLOSS environments through empirical analysis of FLOSS data. This can be accomplished through extracting information from FLOSS repositories in order to identify patterns, progress, evolution and achievements within these participatory groups. Our methodology proposes the use of Process Mining techniques in order to carry out the analysis.

From their immense work on workflow logs and Process Mining, Van der Aalst et al. [39] highlight two fundamental reasons for Process Mining on Event Logs that directly relate to our work. The first purpose that the method serves is to discover how people and procedures really work. An example could be the understanding of flow of patients in a hospital. While information about activities in these environments is available, the actual underlying process is lacking. The second reason is the use of Process Mining for Delta analysis or conformance checking. This implies a comparison of results from the actual process models discovered from Event Logs with some predefined or pre-existing models also referred to as a-priori models.

With the existence of descriptive or prescriptive models that specify how procedures are assumed to occur or how people in organizations are expected to work, comparison with the results of the actual discovered process models will help in detecting discrepancies and more importantly provide insights on how the real learning behavior occurs in FLOSS environments and enrich this area of research.

In the reminder of the thesis, we detail how our approach is implemented following the structure given in the next section.

1.6 Structure of the Thesis

This thesis has been structured in 10 concise chapters in order to present the results of our experiments in the most effective possible way.

— Chapter 1: Introduction and Related Work (Proposal)
In this chapter, we set the scene for our research by introducing the topics, identify the objectives of the study, defining hypotheses and describing our proposed problem solving approach.

— Chapter 2: Mining FLOSS repositories: State-of-the-Art (tools and techniques)
This chapter reaffirms the need for our proposed approach in realizing the objectives of this study by highlighting the importance of mining FLOSS repositories, reviewing current mining techniques and showing the lack of a systematic technique to mine FLOSS repositories to provide empirical evidence of learning processes.
— **Chapter 3: Proposed Approach**
In this third chapter, we tackle the backbone of our proposed approach. We implement the first two steps of the proposed methodology and define an ontology for learning processes in FLOSS (OntoLiFLOSS) and develop our a-priori models in the form of Workflow nets.

— **Chapter 4: Process Models for learning process: Preliminaries**
In this chapter, we identify and describe the fundamentals for mining FLOSS repositories and constructing Event Logs. In essence, we exploit potential text mining techniques and tools one can use to analyze FLOSS data and propose the use of Semantic Search based on Catalogs. This chapter is the implementation of step 3 of our methodology.

— **Chapter 5: Process Models for learning process: Technical Specifications**
We complete step 3 of the methodology and start step 4 by providing the technical ingredients (algorithms) used to generate the Event Logs. These Event Logs are then used for process discovery as described in chapters 6 – 8.

— **Chapter 6: Process Maps for Initiation Phase: Empirical Results**
This chapter starts a series of three chapters that describe the generation of process maps and contextualize these empirical results in light of our hypotheses according to step 5 of the methodology. Specifically, the part we deal with in this chapter pertains to the Initiation Phase of the learning process.

— **Chapter 7: Process Maps for Progression Phase: Empirical Results**
The focus of this chapter is on describing the process discovery endeavor during the Progression phase of the learning process.

— **Chapter 8: Process Maps for Maturation Phase: Empirical Results**
The process discovery step is concluded in this chapter with details of empirical results about the Maturation phase of the learning process. We give an overall impact of this process on our objectives and discuss the transition to next step.

— **Chapter 9: Validation of Learning Process Models: Conformance Checking**
The core of this chapter pertains to the application of conformance analysis techniques to validate process models. This validation entails the use of two metrics, fitness and appropriateness, in order to discuss the similarities and discrepancies or the lack thereof between the a-priori models discussed in Chapter 3 and the final process models discovered from Event Logs.

— **Chapter 10: Conclusion and Future Work**
This chapter concludes our research efforts with a summary of our findings, the validation of our hypotheses (step 7) and definition of possible future work.
CHAPTER 2: MINING SOFTWARE REPOSITORIES: STATE OF THE-ART

2.1 Introduction
The phenomenon of FLOSS has undoubtedly triggered extensive research endeavors. At the heart of these initiatives is the ability to mine the data with the hope to provide empirical evidence to existing questions on FLOSS. Needless to say this work is part of supporting quantitative claims of learning processes with empirical traceable data-driven evidence. Hence, mining FLOSS repositories is the driving force in proving any theories about the data they store. The Mining Software Repositories (MSR) series encompasses annual workshops, conferences and academic gatherings dedicated to finding ways of analyzing FLOSS repositories such as version control repositories, mailing list archives, bug tracking systems, issue tracking systems, etc. [175].

In this chapter, it is critical that we explore some of the state-of-the-art techniques and activities for mining FLOSS repositories before we discuss our approach in the next chapters. Kagdi et al. [56] produced an extensive survey that is quite expressive of critical milestones reached as part of Mining Software Repositories. Here, we succinctly present a few attempted approaches that converge with the objectives of our endeavor. We consider these approaches in terms of the type of software repositories to be mined, the expected results guiding the process of mining as well as the methodology and techniques used herein.

Software repositories in this context include Concurrent Versions System (CVS), requirements/bug-tracking systems as well as communication archives (MailingLists). They contain a wealth of information about software development processes and activities over the period of software existence. Additional information that can be in these repositories includes system or software versions, changes and details of changes.

In this chapter, we briefly explore these approaches and review some of the tools developed in recent years for mining FLOSS repositories.

2.2 Mining Software Repositories: Leading factors
Investigating software repositories is guided by following a number of factors. Kagdi et al. [56] highlight four factors according to which software repositories are investigated. These include information sources, the purpose of investigation, the methodology as well as the quality of the output.

Regarding information sources, it is safe to mention that we have repeatedly stressed and named some of them in this thesis. Additionally, literature on MSR enumerate source-control systems, defect-tracking systems and archived communications as the main sources of data utilized while conducting investigations in FLOSS [56]. Source-control systems are repositories for storing and managing source code files in FLOSS. Defect-tracking systems, as the name suggests, manage bugs and changes reporting while archived communications encompass message exchanges via email in discussion groups and forums between FLOSS participants.

The next critical element at the heart of MSR investigation is the purpose. This is at the start of any research endeavor. It defines the objectives and produces questions whose answers are sought after during
the investigation. In our instance, the purpose is to mine these volumes of data in order to establish to what extent learning processes are traceable through knowledge exchange during day-to-day FLOSS activities.

After identifying the sources, determining the purpose, the next step is to decide on the methodology for mining the data and answering the questions. Due to the nature of investigative questions, available approaches present in the literature revolve around defining and setting some metrics that can be used to verify certain properties on the extracted data. For example, some metrics for assessing software complexity such as defect density and extensibility can be verified on different versions of submitted software in SVN over a period of time deduce properties that explain some form of software evolution. In our case, the three identified data sources are to be extensively parsed following a developed ontology used as a canvas for learning processes identification in FLOSS activities. The key lies in classifying contributors in two important classes of learning participants: Novices or Experts. A step further will be exploring the traceability of these two groups’ knowledge exchange activities over a certain period of time.

The last factor paramount to the investigation of FLOSS repositories is evaluation. Evaluating hypotheses as formed based on constructed investigative questions. In the context of software evolution for example, in order to evaluate FLOSS repositories two information retrieval assessment metrics can be used. These include precision and recall on the amount of information used as well as its relevance. In our context, in order to evaluate our approach in mining FLOSS repositories, we conduct a conformance analysis on a-priori learning models based on the data retrieved from these repositories. This is explored in details in the next chapters.

2.3 Mining Techniques: Selected, Relevant Approaches

This section explores some of the approaches as reported in the literature pertaining to the investigation of software repositories as guided by these leading factors. We have tried to look at approaches that might have a direct link with our work so as to identify similarities in the essence of our work and hence choose an appropriate or combination thereof for our endeavor.

2.3.1 Bug Fixing Analysis

The first chosen approach for illustration pertains to mining software repositories through analyzing bug fixing in FLOSS. Śliwerski et al.[57] present some results on their investigation on how bugs are fixed through introduced changes in FLOSS. The main repositories they used are CVS and Bugzilla along with the relevant metadata. While the purpose of their work was to locate changes that induce bug fixing by coupling a CVS to a BUGZILLA, our interest is to describe the methodology they used to investigate these repositories. Their methodology can be summarized in these three steps:

1. Starting with a bug report in the bug database, indicating a fixed problem.
2. Extracting the associated change from the version archive, this indicates the location of the fix.
3. Determining the earlier change at this location that was applied before the bug was reported.

The authors argue that, after the earlier change has been determined, one can conclude that this earlier change caused the later fix and hence, it is referred to as a fix-inducing change. In our instance, this procedure sheds some guiding lights on how to extract data from version and bug archives as well as how the linkage between bug reports and changes can be established. During the course of this procedure, in order to mirror the CVS, a number of techniques were used. Some of these techniques are detailed by Zimmermann and Weibgerber [58].
Step 1 is to identify fixes. This is done on two levels: syntactic and semantic levels. At the syntactic level, the objective is to infer links from a CVS log to a bug report while at the semantic level the goal is to validate a link using the data from the bug report [57]. Practically, this is carried as follows.

Syntactically, log messages are split into a stream of tokens so as to identify the link to Bugzilla. The split generates one of these items as a token [57]:

- a **bug number**, if it matches one of the following regular expressions (given in FLEX syntax):
  - bug[# \t]*[0-9]+,
  - pr[# \t]*[0-9]+,
  - show\_bug\_cgi\_id=[0-9]+, or
  - \[[0-9]+\]
- a **plain number**, if it is a string of digits [0-9]+
- a **keyword**, if it matches the following regular expression:
  - fix(e[ds])?|bugs?|defects?|patch
- a **word**, if it is a string of alphanumeric characters

A syntactic confidence syn of 0 is assigned to a link and its confidence raised by 1 if the number is a bug number and if the log message contains a keyword, or the log message contains only plain or bug numbers. For example, these log messages are considered [57]:

→ Fixed bug 53784: .class file missing from jar file export
The link to the bug number 53784 gets a syntactic confidence of 2 because it matches the regular expression for bug and contains the keyword fixed.

→ 52264, 51529
The links to bugs 52264 and 51529 have syntactic confidence 1 because the log message contains only numbers.

Furthermore, the role of the semantic level in Step 1 of the methodology is to validate a link \((t, b)\) by taking information about its transaction \(t\) and checking it against information about its bug report \(b\). A semantic level of confidence is thus assigned to the link based on the outcome. This is raised accordingly and incremented by 1 following a number of conditions such as the bug \(b\) has been resolved as **FIXED at least once**, the short description of the bug report \(b\) is contained in the log message of the transaction \(t\). Two examples on ECLIPSE are shown as follows:

→ Updated copyrights to 2004
The potential bug report number “2004” is marked as **invalid** and thus the semantic confidence of the link is 0.

→ Support expression like \((i)+= 3;\) and new \(\{1\}[0]\) + syntax error improvement
“1” and “3” are (mistakenly) interpreted as bug report numbers here. Since the bug reports 1 and 3 have been fixed, these links both get a semantic confidence of 1.

The rest of the process (Step 2 and 3) involves a number of tasks ranging from manual inspections of returned links to eliminate links that do not satisfy this condition:

\[
\text{sem} > 1 \lor (\text{sem} = 1 \land \text{syn} > 0)
\]
Figure 13. Rigorous manual inspection of randomly selected links for Bug fixing [57]

Figure 13 shows that this process involves rigorous manual inspection of randomly selected links that are to be verified based on this condition.

After applying this concept on ECLIPSE and MOZILLA with respectively 78,954 and 109,658 transactions for changes made until January 20, 2005, Śliwerski et al. [57] presented their results for 278,010 and 392,972 individual revisions on these projects respectively. Some of these results concern the average size of transactions for fixes in both projects, the different days of the week during which most changes are projected to occur etc.

2.3.2 Software Evolution Analysis

The second approach was conducted by German [59] to present the characteristics of different types of changes that occur in FLOSS. In this study, the CVS and its related metadata are used as information sources. The collective nature of software development in FLOSS environments allows for incremental changes and modifications to software projects. These progressive changes can be retrieved from version control systems such as CVS or SVN and parsed for analysis. In this approach, the author investigated changes made to files as well as the developers that mostly commit these changes over a period of time. His argument also suggests that analyzing the changes would provide clarifications on development stages of a project in light of addition and update of features [59].

A number of projects considered for this purpose include PostgreSQL, Apache, Mozilla, GNU gcc, and Evolution. Using a CVS analysis tool called softChange, CVS logs and metadata were retrieved from these projects for this investigation. A new algorithm called Modification Records (MRs) is proposed and the author claims the algorithm provides a fine-grained view of the evolution of a software product. Noticeable from the work is the methodology used for data mining the chosen repositories. The first step was to retrieve the historical files from CVS and rebuild the Modification Records from this info as they don’t appear automatically in CVS. SoftChange, through its component file revision, makes use of sliding window algorithm heuristic to help organize this information.

Briefly explained, the algorithm takes two parameters ($\delta_{\text{max}}$ and $T_{\text{max}}$) as inputs. $\delta_{\text{max}}$ depicts the maximum length of time that an MR can last while $T_{\text{max}}$ is the maximum distance in time between two file revisions. The idea is that a file revision is included in a given MR on the basis of these conditions [59]:
Chapter 2: Mining Software Repositories: State-of-The-Art

2.3.4 Identification of frequently occurring changes (FACs) in source files

Another attempt at investigating FLOSS repositories is conducted by Van Rysselberghe and Demeyer [60]. The purpose of this study was to make use of the clone detection methods on source code in CVS as well as its metadata in order to analyze frequently occurring changes (FACs) in source files. The idea is to document changes occurring in FLOSS using a technique tailored in a similar manner to the standard concept of frequently asked questions or FAQs [60]. The rationale of FAQs is to gather some basic questions and answers that are representative of frequent questions and corresponding answers in order to reduce the continual posting of the same basic questions. The objective is to identify frequently applied changes (FACs) since these changes record general solutions to frequent and recurring problems.

This technique requires the use of a repository that stores changes on source code. Therefore, CVS is appropriate as it records all changes made over the software life cycle. In this instance, a frequently applied change can be understood as a change to the code which occurs multiple times in the evolution of a system [60]. Using proper CVS commands such as some “cvs log” and “cvs diff” commands, change data can be extracted. These data include the difference in code before and after the change, the date and time of the change, the file involved. With this information obtained, the next step is to parse this information and identify FACs. Locating FACs implies locating similar code fragments and this can be done by applying clone detection techniques.

These techniques are developed to help identify duplicated or cloned code fragments in a program source code. During such process, a tool called CCFinder was used to analyze text-file containing codes with FACs as retrieved using clone detection techniques. Based on some threshold values, the study asserts that high threshold values allow the identification of recurring and product specific changes while low threshold values lead to the identification of frequently applied generic changes.

Using Tomcat as a case study, observations drawn from the initial experiment include for instance that FACs identified with a high threshold and specific to one product can be used to study and understand the
motivation and success behind an applied change. Also, the removal of a recently added code fragment may give an indication for the reasons behind success or failure of changes in general. On the other hand, FACs with a low threshold can help in deriving low maintenance strategies automatically.

In this study, as in the previous one, no specific details are given with regards to the step-by-step execution of the technique in investigating CVS. Nevertheless, a general idea is provided as well as the obtained results. This work exemplifies how FLOSS repositories can be explored and investigated for a number of reasons as well as the mounting interest in for the understanding FLOSS repositories.

### 2.3.5 Identification of developers’ identities in FLOSS repositories

The next case of FLOSS investigations is about the identification of developers identities in FLOSS repositories. Given the dynamic nature of developers’ behaviors in adopting different identities in FLOSS, the task of identification becomes cumbersome. Nevertheless, one solution in this regards has been to integrate data from multiple repositories where developers contribute. Sowe and Cerone [24] proposed a methodology making use of repositories from the FLOSSMetrics project in identifying developers who make contributions both by committing code to SVN and posting messages to mailing lists. The methodology was summarized in Section 1.2.1.4 (See also Figure 5).

Robles and Gonzalez-Barahona [61] conducted a similar study based on the application of heuristics, to identify the many identities of developers. The methodology was applied taking the GNOME project as a case study. The key feature in both of these endeavors in this regard is to identify and merge identities corresponding to the same and unique developer. Hence, the first step consists in identifying the possible types of identities one can adopt in order to use the many FLOSS repositories. Some of these include: one or more email addresses on mailing lists; in a source file a developer can be identified with a real file name, a nickname, an email address or an RCS-type identifier; in a versioning system the user can be identified with an account that is recorded on a log file while making use of an account associated with an email address for a bug tracking system. The next step is about classifying these identities into primary and secondary [61].

Primary or mandatory identities are the ones used as required for access to mailing lists, CVS and bug tracking systems while secondary or redundant identities appear usually with primary identities even though, many times, the same user can use different identities for accessing the same repository [61]. The next step is to locate these identities from repositories using some heuristics and filling the identity tables before matching. It is inferred that real names can sometimes be retrieved from different email addresses and thus stored in the Matches table in order to allow the matching process based on some form of heuristics with manual human inspection involved as well. Therefore, the matching is based on prerequisite information obtained in the second step as depicted in Figure 14.
One of the core tasks is how to match all the identities that correspond to the same actor. This is facilitated by populating the Matches table with as much (and as accurate) information as possible. Heuristics, as hinted, are used to this end. Some of them include [61]:

- In many cases it is common to find a secondary identity associated to a primary one. Often in mailing lists, source code authorship assignments and bug tracking system. The authors argue that in such cases, the primary identity (usually an e-mail address) may have a `real life' name associated with it;
- Sometimes an identity can be built from another one whereby a developer's real name is extracted in some cases from the e-mail username;
- In many cases one identity is a part of some other. For example, in some instances it is possible that the username as used for CVS is the same as the username part of the user's email address;
- Finally, it is common to find some projects or repositories that maintain specific information useful for matching. Projects like KDE or platforms like Sourceforge.net maintain files containing lists, for every person with write access to the CVS, his real life name, his username for CVS and an e-mail address.

These steps can be summarized as seen in Figure 15. Robles and Gonzalez-Barahona [61] applied this methodology to the GNOME project where 464,953 messages from 36,399 distinct e-mail addresses were fetched and analyzed, 123,739 bug reports, from 41,835 reporters, and 382,271 comments from 10,257 posters were retrieved from the bug tracking system. Around 2,000,000 commits, made by 1,067 different committers were found in the CVS repository. The results showed that 108,170 distinct identities could be extracted and for those identities, 47,262 matches were found, of which 40,003 were distinct (with the Matches table containing that number of entries). Using the information in the Matches table, 34,648 unique persons were identified.
2.3.6 Identification of specific lines of codes in CVS

Another interesting work on mining FLOSS repositories we have chosen for convenience was conducted by Yao [62]. In this endeavor, the objective is to search through source code in CVS and related metadata to find lines of code in specific files etc. This is done through a tool called CVSSearch introduced in Section 2.4. The technique or mining approach used here to analyze CVS comments allows to automatically map the commit comment and the lines of code that it refers to. This is useful as CVS comments provide additional information that one cannot find in code comments. For instance, when a bug is fixed, information about this does not typically appear in code comments but can be found in CVS. And as part of investigating FLOSS, one can search for code that is bug-prone or bug-free based on CVS comments where these lines of code can be referenced.

Hence, this technique entails searching for lines of code by their CVS comments in producing a mapping between the comments and the lines of code to which they refer [62]. This is carried as follows [62]:

- Consider a file $f$ at version $i$ which is then modified and committed into the CVS repository yielding version $i+1$.
- Also, suppose the user entered a comment $C$ which is associated with the triple $(f, i, i+1)$.
- By performing a diff between versions $i$ and $i+1$ of $f$, it is possible to determine lines that have been modified or inserted in version $i+1$, the comment $C$ is thus associated with such lines.
- Additionally, in order to search for the most recent version of each file, a propagation phase is undertaken. During this phase, the comments associated with version $i+1$ of $f$ are “propagated” to the corresponding lines in the most recent version of $f$, say $j \geq i+1$. This is done by performing diffs on successive versions of $f$ to track the movement of these lines across versions until version $j$ is reached.

Ying et al. [65] take a different perspective to investigate source code. Using the source code in CVS, the authors propose an approach to study communication through source code comments using Eclipse as a case study. This is premised on a principle of good programming that asserts that comments should “aid the understanding of a program by briefly pointing out salient details or by providing a larger-scale view of the proceedings” [65]. As part of understanding FLOSS activities, it has been found that comments in these environments are sometimes used for communication purposes. An example of a comment such as
“Joan, please fix this method” addresses a direct message to other programmers about a piece of code but it is usually located in a separate archive such as CVS.

Hence, Ying et al. [65] provide an informal empirical study on Eclipse task comments in Java source code to analyze commenting on source code. Tasks comments simply refer to the use of task tag strings such as “TODO” embedded in the source code in order to browse a summary of the places in the code with a comment that contains a task tag [65]. Considering the IBM internal codebase, called the Architect’s Workbench (AWB), with a team of less than ten developers on data retrieved from the AWB CVS repositories on February 9, 2005, it has been shown that the codebase consisted of 2,213 files while the code contained 221 task comments [65].

The study concludes that unlike bug report or JavaDoc, task comments are shorter and more informal making it cumbersome and even difficult to process using natural language. Furthermore, it argues that task comments are not always effective, given their structure, nature and usability. Therefore, in order to communicate to other team members in Architect’s Workbench (AWB), a developer may better use a Java comment, straight in the source code rather than a task comment.

2.3.7 Help provision for new FLOSS members and roles identification in FLOSS

An important purpose of mining FLOSS repositories is the provision of adequate information for new developers in FLOSS. Given the dynamic mode of operations in FLOSS, it is quite difficult for newcomers to come up-to-speed with a large volume of data concerning a project he/she has joined. Hence, a new tool called Hipikat is introduced [66-67]. The idea is that Hipikat can recommend to newcomers key artifacts from the project archives. Basically, this tool is assumed to keep an implicit group memory from the information stored in a project archives, and based on this information, it provides information to new developers as needed. The Eclipse open-source project is used as a case study in applying this approach.

The building blocks of this approach are twofold. Firstly, an implicit group memory is formed from the artifacts and communications stored in a project history. Secondly, the tool presents to the new developer artifacts as selected from this memory in relevance to the task being performed. A group memory can be understood as a repository used in a FLOSS work group to solve present needs based on historical experience. In essence, the purpose of Hipikat is to allow newcomers to learn from the past by recommending items from the project memory made of source code, problem reports, newsgroup articles, relevant to their tasks [66].

This model depicts four types of artifacts that represent four main objects that can be found in FLOSS projects as shown in Figure 16. Change tasks (tracking and reporting bugs like in Bugzilla), source file versions (as recorded in CVS), mailing lists (messages posted on developer forums) and other project documents like requirements specification and design documents. An additional entity called Person is included to represent the authors of the artifacts.
The last attempt considered is the work on developer roles and contributions done by Huang and Liu [68]. Similar to numerous other studies available in the literature, this is based on a quantitative approach to analyze data in FLOSS. Using the CVS as the experimental repository, a network analysis is performed in order to construct social network graphs representing links between developers and different parts of a project [68]. Standard graph properties are computed on the constructed networks and thus an overview in terms of developer’s activities is given to explain the fluctuations between developers with lower and higher degree.

2.4 Tools

Tools are central to the sheer of work done with the purpose of mining software repositories. A number of tools have been developed throughout this process, and we look at some of them to describe the aspects of software repositories they can analyze. Some of these tools include CVSSearch, MLStats, CVSAnaly, CVSGrab and SoftChange.

**CVSSearch:** This tool is used for mining CVS comments [62]. As the authors argue, the tool takes advantages of two characteristics of CVS comments. Firstly, a CVS comment more likely describes the lines of code as involved in the commit; and secondly, the description given in the comment can be used for many more versions in the future. In other words, CVSSearch allows one to better search the most recent version of the code by looking at previous versions to better understand the current version [62]. The technique the tool illustrates has been described in Section 2.3.6. The tool, thus, is an implementation of the algorithm.

**CVSgrab** is a visualization tool whose objective of the tool is to visualize large software projects during their evolution [69]. CVS query mechanisms are embedded in the tool to access CVS repositories both
locally and over the internet. Using a number a metrics, CVSgrab is able to detect and cluster files with similar evolution patterns [69]. One of the key features is its particularity to interactively show evolutions of huge projects on a single screen, with minimal browsing. The tool’s architectural pipeline is given in Figure 17.

![Figure 17. CVSgrab’s architectural pipeline [69]](image)

As output, CVSgrab uses a simple 2D layout where each file is drawn as a horizontal strip, made of several segments. The x-axis encodes time, so each segment corresponds to a given version of its file. Color encodes version attributes such as author, type, size, release, presence of a given word in the version’s CVS comment, etc. Atop of color, texture may be used to indicate the presence of a specific attribute for the considered version. File strips can be sorted along the y-axis in several ways, thereby addressing various user questions [69].

**SoftChange** The purpose of this tool is to help understand the process of software evolution [70]. Based on analyzing historical data, SoftChange allows one to query who made a given change to a software project (authorship), when (chronology) and, whenever available, the reason for the change (rationale). Three basic repositories are used with SoftChange for analysis. CVS, bug tracking system (Bugzilla), and the software releases [70]. The tool’s architecture is represented in Figure 18.

![Figure 18. SoftChange Architecture [70]](image)
**MLStats** is a tool used for mailing list analysis. The purpose of the tool is to extract details of emails in this repository. Data extracted from messages depending on from senders, receivers, topics of message and time stamps as associated with the exchanged emails [72-73]. The tool makes use of the email headers to derive the analysis.

The last tool we explore is the **CVSAnalY**. This is a CVS and Subversion repository analyzer that extracts information from a repository. Embedded with a web interface, it outputs the analysis results and figures that can be browsed through the interface [71]. Specifically, CVSAnalY analyses CVS log entries that represent committers’ names, date of commit, the committed file, revision number, lines added, lines removed and an explanatory comment introduced by the committer [71]. Three main steps are part of this approach: preprocessing, database insertion and post-processing [71].

The preprocessing step includes downloading data from the CVS repository and removing aggregated modules in order to avoid multiple commits count. Once this is done, the logs are retrieved and parsed in order to ensure that the obtained log files are transformed into a more structured format e.g. SQL for databases or XML for data exchange. At completion of this step, the data is stored into a database after computing some summarization and database optimization information. In the post-processing step, a number of scripts are run to interact with the database. These scripts provide statistical information about the database, compute several inequality and concentration indices and generate graphs for the evolution in time for parameters such as number of commits, number of committers etc. as needed [71].

**2.5 Conclusion: Process Mining and FLOSS repositories**

FLOSS repositories store a sheer volume of data about participants’ activities. A number of these repositories have been mined using some of the techniques and tools we have discussed in this paper. However, to the date, there has not been any concrete investigation into how logs from FLOSS repositories can be process mined for analysis. This may be attributed partly to two apparent factors. Firstly, researchers interested in mining software repositories have not come across process mining and thus its value is unexploited; secondly, the format of recorded in FLOSS poses a challenge in constructing Event Logs.

Nevertheless, after reviewing existing mining techniques and the analysis they provide on the data, one can infer the type of input data, the expected output and thus construct logs that can be used for analysis through any of process mining recognized tools such as the ProM framework or Disco.

We present an approach that shows a logical flow of steps required to construct Event Logs from FLOSS data. In the following chapters, we start exploring and unpacking this approach whose core element is Process Mining in order to trace and analyze learning processes in FLOSS repositories such as Mailing archives, Bug reports etc.
CHAPTER 3: PROPOSED APPROACH

3.1 Introduction

In Chapter 1, we introduced and briefly discussed our proposed approach to mining FLOSS repositories in order to conduct the study on the existence of learning processes or lack thereof in these repositories. In this approach, we propose to follow a non-sequential seven-step conceptual framework that explains the path to solving our problem. The adoption and use of Process Mining techniques is the core of this approach. All these steps are conjured in order to steer the construction of Event Logs, the development of corresponding process models and their evaluation in light of our objectives.

The steps include developing an appropriate terminology or ontology for learning processes in FLOSS, contextualizing learning processes through a-priori models, generating Event Logs, generating corresponding process models, interpreting and evaluating the value of process discovery, performing conformance analysis and verifying our formulated hypotheses with regard to tracing learning patterns in FLOSS communities.

A certain level of iteration is to be maintained in the execution of this framework. This means that the steps do not necessarily have to be performed or undertaken strictly in a sequential fashion; however as the needs arise, one can still go back and forth in order to update or adjust the deliverables as required.

In this chapter we unpack the first two steps of our approach to the proposed solution. The remaining steps are tackled accordingly in the subsequent chapters.

3.2 Step 1: Ontology for Learning Processes in FLOSS (OntoLiFLOSS)

In step 1, the idea is to design and agree on a common definition of concepts and activities that one can use to identify whether or not learning has taken place during an interaction between FLOSS members. These concepts will guide the rest of the process in that they will help trace a learner's and teacher’s activities trail in mailing list, SVN, bug trackers and even forums in order to understand the occurrence of learning processes during such exchanges. One way of achieving this is through ontology.

Given that our core objective is to find empirical evidence of learning traces among participants in FLOSS repositories, we need a sort of guideline indication that provides a “generic” representation of the structure of information and conceptualization of knowledge pertaining to learning processes in FLOSS repositories for a given FLOSS project. This can be achieved by means of Ontologies due their preponderant role in knowledge representation.

In describing the role of ontologies in computer science, Fonseca supports that ontology is an engineering artifact that is constituted by a specific vocabulary used for the purpose of describing a specific reality or domain [74]. Ontologies can also be useful for the validation of conceptual models and conceptual schemas [74]. Wilson adds to this role by suggesting that ontologies “attempt to formulate a thorough and rigorous representation of a domain by specifying all of its concepts, the relationships between them and the conditions and regulations of the domain” [75].

Furthermore, ontologies play a significant role in software engineering. Happel and Seedorf [76] advocate the adoption of ontologies to help the communities of Software Engineering and Knowledge Engineering make use of common topics and concepts. As members of development teams tend to conceive topics differently, this creates a disparity with regard to the own understanding of central concepts, making it difficult for them to communicate efficiently on the concepts of the same domain. Hence, the use of
ontologies in various stages of the development lifecycle is critical as it can alleviate this gap by providing common grounds and vocabularies given their potential for knowledge representation and process support [76-78].

In Open Source, the adoption of ontologies is paramount. With millions of users converging on the same concepts and topics, a lack of common knowledge representation would be chaotic. Few attempts can be observed in [77, 79]. Mirbel [77] introduces and describes the OFLOSSC (An Ontology for Supporting Open Source Development Communities) as an extension to the previous OSDO (Ontology for Open Source Software Development) presented by Simmons and Dillon [79]. Tifous et al. [80] introduce an ontology that specifies open source software environments as communities of practice from which Mirbel [77] borrows a few guidelines as well. While these ontologies describe classes and properties for participants as well as roles of individuals in Open Source environments, their scope of knowledge representation describes common concepts one needs to understand these communities.

For the purpose of our analysis, the focus is on learning processes in these communities. Hence, the premise of this task is predicated on the established assumption that in FLOSS communities, members engage in processes of knowledge exchange that can be regarded as learning processes. In order to explain how this takes place, we identify all relevant activities FLOSS members engage in and on this basis, develop the ontology. A number of studies have been critical in this instance [1-13] and specifically the works conducted by Cerone [11] as well as Sowe and Cerone [27] provide a lot of grounds for the identification of terms and concepts one can use to identify learning activities, participants and related classes in FLOSS.

Through an iterative process of ontologies design, the objective is to formalize and represent knowledge structures for the purpose of using them as a roadmap to understanding crucial learning resources and concepts that can be found in FLOSS repositories.

### 3.2.1 Methodology

A wide range of methodologies exist as guidelines for the conceptualization and design of ontologies [81-83]. For simplicity and user-friendliness, we have adopted a short methodology to design our ontology. Based on the immense information and resources pertaining to FLOSS environments available in the literature that we have explored, we have designed the ontology following a top-down approach comprising the following five steps:

a. **Information Collection:**
   Our sources of information for the building of the ontologies are mainly studies on FLOSS environments in the literature [1-13, 27] as well as generic assumptions about learning.

b. **Concepts Identification and classes definition:**
   Based on the availability of a plethora of materials on activities in FLOSS, we have defined some concepts and relations for the ontology to represent entities, resources and constraints of learning in FLOSS environments.

c. **Definition of Class Taxonomy:**
   This helps in specifying and defining classes with their subclasses

d. **Properties and labels definition:**
   Properties give an indication of classes attributes as well as their connectivity.

e. **Ontology Formalization**
   The language we chose for our ontology formalization is OWL-DL given its large-scale semantic web support. We make use of Protégé and OntVis to generate the graphical representations of our classes in the OntoliFLOSS.
Our Ontology is called “OntoLiFLOSS”. This is an acronym for Ontology of Learning in FLOSS. The nine learning processes we have identified can be succinctly demarcated from each other using the initiation activities, the initiator, learning content or topic as well as the evidence of learning through contribution and activities performed by the learner. Although we are not able to fully assert that there are empirical traces of all the processes in CVS, mailing archives and bug reports, we think that the knowledge structural representation in the form of ontology may trigger further investigations and even additional research directions in FLOSS. Furthermore, we have chosen two main tools for the implementation and visualization of the ontology: Protégé 4.3 and Knoodl-OntVis. The former previews main classes as well as their subsequent subclasses while the latter helps in building graphs with relevant connecting properties between the classes.

3.2 OntoLiFLOSS: Main Concepts

Based on the FLOSS information as obtained from the literature and given the purpose of our study, we have assumed that the ontology for learning processes in FLOSS called “OntoLiFLOSS” is made up of 138 entities (expandable as required) and detailed with the following main building blocks:

**Classes (80):** These classes are representation and classification of information on learning processes that are supported by a particular FLOSS project through performing a certain number of activities referred to as “learning activities”. These activities are carried out by participants that can be either Experts or Novices with regards to their involvement in the learning process and can be organized into Teams. A number of resources are used to support the process that can be tracked through FLOSS repositories to which inputs and learning outputs are committed.

**Object Properties (38):** These are associations or relationships that explain the link between the different classes/concepts as described above.

**Data Properties (5):** The data properties are attributes mainly for participants, whether they are novices or experts, which document their competency level, their experience, level of contributions as well as their skillset.

**Annotation Properties (1):** This is just an explanatory comment on the ontology.

**Individuals (9):** These are 9 instances of possible learning processes that can be reconstructed considering all the classes in the ontology.

**Datatypes (3):** These describe the data type for the 5 data properties as indicated above.

3.2.2.1 Classes

Of the total of 80 classes in OntoLiFLOSS, 10 classes are super classes that can be expanded to identify subclasses at the appropriate granularity as needed. Figures 19 and 20 give an abstract representation of these classes as well as their connecting associations.
We now give a detailed description of these 10 super classes as well as their subclasses with regard to the direct links they create with other classes.
**FLOSSProject Class** This class depicts any given project used in the investigation or evaluation of FLOSS environments. The instances of the class can be typical projects from Sourceforge.net or GitHub or any other FLOSS community platforms such as KDE, NetBeans or any other project of convenience.

![Diagram of FLOSSProject Class](image)

**Figure 21. Network graph of the FLOSSProject Class, immediate subclasses and related classes**

Figure 21 reflects the direct neighborhood for the FLOSSProject Class that is comprised of 8 main classes. It gives a full visualization graph of the neighborhood for concepts and their related associations (through object properties).

**LearningActivity Class** This class depicts concepts about all activities that are directly involved with the learning process. Six classes constitute the subnet or neighborhood for the LearningActivity class as shown below. The class also has three subclasses that classify the types of learning activities as seen in Figure 22. The class also has three subclasses that classify the types of learning activities as depicted in Figure 22.
The three subclasses that depict the different stages of learning include Initiation, Progression and Maturation.

**Initiation** Two main activities (subclasses) characterize and are part of this stage: Observation and ContactEstablishment.
self-explanatory subclasses such as IdentifyExpert, FormulateQuestion, PostQuestions, ReadMessages, ReadPost, ReadSourceCode, and CommentPost.

In ContactEstablishment, the focus of the representation is on the steps that any learning participant (Novice or Expert) undertakes to establish a contact between the actors and to initiate the actual learning partnership. This happens through three activities: ContactExpert, ContactNovice and SendDetailedRequest.

**Progression** In this stage, the ontology defines three subclasses: Revert, Post and Apply. Each of them further branches out with several subclasses as depicted in Figure 24:

![Figure 24. Subclasses during Progression Stage](image)

**Revert** This activity (class) encompasses all the steps Novice and Expert go through to provide the required information. Three basic classes or sub-activities occur here: SendReply, where there is a reaction to any attempt of contact established from either the Novice or Expert; ReviewThreadPosts refers to the ability to analyze and react as needed to comments and posts related to a particular content that is the subject of learning; ReviewThreadCode concerns analyzing the code (when applied) and engaging accordingly for a particular topic of interest.

**Post** This is one of the basic activities that express the contributions of the Novice. It involves three activities: PostQuestions, which refers to the ability to ask further questions or comments on more advanced topics; ReportBugs, which entails the ability to scrutinize the source code and run pieces of code to identify potential flaws; SendFeedback, which refers to replying to questions or comments (including reporting identified flaws).
Apply In applying any knowledge or skill as a result of the learning process, the Novice can perform some activities represented as the subclasses of this class. These include: AnalyzeSourceCode, for the ability to review the submitted code and bugs, especially when the piece of code relates to the area in which skills have just been acquired; CommentOnCode, for the ability to comment on the code to show progress or explain the logic behind that part of the software; ReplyToPost, which refers to the confidence to be active on the mailing list and reply to questions or comments pertaining to the same thread or any other topic directly or indirectly linked to the newly acquired skills; ReportBugs, for the ability to report bugs for submitted piece of code or any other version release; RunSourceCode, where, in running a piece of code, the Novice is able to accomplish all the above activities.

Maturation This class of activities identifies the last phase of the learning process, which asserts how the Novice has mastered the skills learnt during the learning process. These activities include as subclasses Analyze, Commit, Develop, Revert and Review, which in turn contain subsequent child classes as shown in Figure 25.

![Figure 25. Subclasses during Maturation Stage](image)

**Analyze** This activity (class) encompasses all the steps Novice and Expert go through to provide the required information or perform requested tasks.

**Commit** With skills growing in a specific area, the Novice becomes confident and can commit some deliverables that can be evaluated and criticized by the community. These activities can be summarized through: SubmitBugReport, which entails the ability to commit any fix or bug report for the interest of the entire community; SubmitCode, which implies submitting some pieces of as part of the project while building reputation for a possible role transition; SubmitDocumentation, through which the Novice submits documentation such as requirements elicitation documents, help documents, user manuals, tutorials etc.
**Develop** This basic activity summarized a set of tasks that the Novice carries out as a result of the skills learnt with regard to software development in FLOSS. These include: *FixBugs*, though which the Novice can identify possible bugs and fix them; *GiveSuggestion*, where the Novice can review peers’ works and provide alternatives when needed, for example what function might be appropriate to perform a particular task etc.; *PostCommentOnCode* refers to the ability to submit comments on the source code for enlightenment; *ReplyToSuggestion*, which entails re-ply and critique suggestions from other Experts or Novices in an active fashion; *WriteSourceCode*, through which the Novice can write and submit source code; *ModifySourceCode*, when the Novice can modify any code and implement suggestion as requested.

**Revert** This is in essence the same activity as in the progress stage. In this class all activities through which the Novice and Expert exchange feedback are represented: *SendReply*, which entails react to any attempt of contact established from either the Novice or Expert; *ReviewThreadPosts*, which implies the ability to analyze and react as needed to comments and posts related to a particular content that is the subject of learning; *ReviewThreadCode*, which signifies analyzing the code (when applied) and provide necessary suggestion if required.

**Review** The Novice and Expert engage in a set of activities to examine the maturity of the learning process. These activities include: *ReviewCommentContents*, in which they actively engage and contribute to comments and posts in the team, about topics in the sphere of the skills acquired and possibly becoming an Expert to a new Novice; *ReviewPosts*, which entails actively engaging and reacting as needed to comments and posts related to a particular content that is the subject of learning; *ReviewSourceCode*, in which they (Novice/Expert) analyze the code (when applied) and engage accordingly for a particular topic of interest.

**Resource Class** This class refers to the resources used as part of learning during development in FLOSS. Such resources include the requirements description documents as well as any documentation needed for the project. Figure 26 depicts the class Resource with its direct neighbors as well as the categories of three subclasses, *DescriptionDoc*, *HelpDoc* and *SourceCode*, which are part of the main resources used in FLOSS that can help identify learning processes.
The three subclasses and their child subclasses, depicted in Figure 27, include DescriptionDoc, HelpDoc and SourceCode and are described as follows.

**Figure 27. Sub classification of the Resource Class**

**DescriptionDoc** This class contains all the documents that provide the description for any activity or stage of the project in the team. The subclasses representing these documents include BugReport, which is a report outlining the description of a found reported bug in a code or piece of software at run time; ProjectRequirementsDesc, which encompasses the documents pertaining to the description of the project, including requirements and all related information regarding the projects operations; UserManuals, which contains the guidelines for the users of the software.

**HelpDoc** This class contains all the documents that provide information for any required help regarding the functionalities of the repositories and projects. These are: FAQ, How-To, and Tutorial.

**SourceCode** This is the content of the Version Control System that contains all the coding done behind any application in FLOSS. It is a major resource of learning as it guides most of the basic activities considered above.

**Repository Class** This is the main class that represents a particular FLOSS repository where learning activities can be observed. Figure 28 depicts the class Repository as well as its neighbors and subclasses. The three subclasses are: VersionControl, where the source code is housed; BugTracker, which contains information about bugs, date of release, and description; MailingList, which represents the contents of online interactions and discussions among participants online in FLOSS.
Figure 28. Network graph of the Repository Class, immediate subclasses and related classes

**Team Class** This is the team of participants, the FLOSS community or forums where participants engage in knowledge exchange. Figure 29 depicts the properties and direct neighbors.

**StandardOfPractice Class** These are rules of engagement that guide the interaction among participants, the usage and licensing of the deliverables in the FLOSS communities. Figure 30 shows the direct neighborhood of classes as well as the different types of practices that can be subcategorized as GNULicense, which represents a fundamental licensing guide for Open Source Software, and PersonalGroundRules, which are rules established and belonging to a given FLOSS Community.

Figure 29. Network graph of the Team Class and related classes
Participant Class This class represents the participant of the learning process. The neighboring classes are connected through associations as depicted in Figure 31. The class has two important subclasses identified as Novice and Expert. These two concepts are critical in understanding and identifying role playing during knowledge exchange activities between FLOSS members. Subclass Novice represents a knowledge requester. This subclass is represented with its neighbors in Figure 32.
With the *Expert* subclass, depicted in Figure 33, the representation refers to the relative knowledge provider at any given point in time during interactions in FLOSS environments.
Figure 34. Graph representation of instances of the LearningProcess Class

Figure 35. Network graph of the LearningProcess Class and related classes
**LearningProcess Class** This class is the main focus and the reason of OntoLiFLOSS. It characterizes the learning process occurring in FLOSS, the other classes in the ontology complete the need to define its semantic conceptualization. Through a set of activities, by means of some resources, the ontology can express, to some extent, learning processes taking place between participants. In Figure 34, we show 9 possible learning processes that one can reconstruct taking into account all the classes in the ontology. We give a more or less complete representation graph of the class and its neighbors in Figure 35. The relationships between the connecting neighbors are given accordingly.

**ProjectRole Class** This subclass represents the basic roles any participants can fulfill in the FLOSS community. We consider the roles identified by Cerone [1]: Observer, PassiveUser, ActiveUser, Developer, and CoreDeveloper. The relationships between the connecting neighbors forming the network are given in Figure 36.

![Figure 36. Network graph of the ProjectRole Class, subclasses and related classes](image)

**LearningStage Class** The objective of this class is to represent the different stages of learning that participants go through during a learning process. Hence, the LearningStage class clearly relates the performed activities to the different stages of learning. Three stages can be identified relating to three layers of activities as in the LearningActivity class: Understanding, Practicing and Developing. Figure 37 depicts a different version of the representation with the subclasses Understanding, Practicing and Developing being depicted as respectively equivalent to the following subclasses of the LearningActivity class: Initiation, Progression and Maturation, with appropriate relationships.
Figure 37. Network graph of the LearningStage Class, subclasses and related classes
3.2.2.2 Properties

Properties are ontology representations of concepts that establish links (relationships) between classes and form networks. Two main types of properties included in OntoLiFLOSS include Object and Data Properties.

About 38 Object Properties summarize the relationships and links between the different classes as seen in Table 3. Five data properties are representative of the main attributes of Participant (either Novice or Expert) relevant with learning. OntoLiFLOSS represents concepts for Experience, Skill Set (acquired through contribution), Contributions (expressed through activities), Competency (built with experience) and Knowledge (acquired through learning).

<table>
<thead>
<tr>
<th>Property/Relationships: Associations between classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• applySkillsThrough : Novice --&gt; Developing/Maturation</td>
</tr>
<tr>
<td>• assessedThrough : LearningProcess --&gt; Repository</td>
</tr>
<tr>
<td>• canPlayA : Participant --&gt; ProjectRole</td>
</tr>
<tr>
<td>• carriesOut : Participant --&gt; LearningActivity</td>
</tr>
<tr>
<td>• commitsIn : Participant --&gt; VersionControl</td>
</tr>
<tr>
<td>• communicatesThrough : Expert/Novice --&gt; SendDetailedRequest</td>
</tr>
<tr>
<td>• documents : Repository --&gt; LearningActivity</td>
</tr>
<tr>
<td>• fosters : Team --&gt; LearningProcess</td>
</tr>
<tr>
<td>• goesThrough : Participant --&gt; LearningStage</td>
</tr>
<tr>
<td>• hasResource : FLOSSProject --&gt; Resource</td>
</tr>
<tr>
<td>• initiatesLearningThrough : Expert/Novice --&gt; Observation</td>
</tr>
<tr>
<td>• isAccompaniedBy : LearningActivity --&gt; Resource</td>
</tr>
<tr>
<td>• isDevelopmentToolFor : Repository --&gt; FLOSSProject</td>
</tr>
<tr>
<td>• isDoneIn : FLOSSProject --&gt; Team</td>
</tr>
<tr>
<td>• isEnabledBy : LearningProcess --&gt; FLOSSProject</td>
</tr>
<tr>
<td>• isGuidedBy : FLOSSProject --&gt; StandardsOfPractice</td>
</tr>
<tr>
<td>• isPartOf : Participant --&gt; Team</td>
</tr>
<tr>
<td>• playedIn : ProjectRole --&gt; FLOSSProject</td>
</tr>
<tr>
<td>• isSupportedBy : LearningProcess --&gt; Repository/Resource/StandardsOfPractice</td>
</tr>
<tr>
<td>• keepsTrackOf : Repository --&gt; LearningActivity</td>
</tr>
<tr>
<td>• learnsThrough : Novice --&gt; Practising/Progression</td>
</tr>
<tr>
<td>• makesContactTo : Novice/Expert --&gt; Expert/Novice</td>
</tr>
<tr>
<td>• monitorsNoviceThrough : Expert --&gt; Revert/Review</td>
</tr>
<tr>
<td>• mustAdhereBy : Participant --&gt; StandardsOfPractice</td>
</tr>
<tr>
<td>• occursThrough : LearningProcess --&gt; LearningStage</td>
</tr>
<tr>
<td>• operatesWith : Participant --&gt; Repository</td>
</tr>
<tr>
<td>• postsAt : Participant --&gt; MailingList</td>
</tr>
<tr>
<td>• producedThrough : LearningProcess --&gt; LearningActivity</td>
</tr>
<tr>
<td>• providesGuidelinesOn : Resource --&gt; Repository</td>
</tr>
<tr>
<td>• reachesOutThrough : Novice --&gt; ContactExpert</td>
</tr>
<tr>
<td>• reachesOutToNoviceThrough : Expert --&gt; ContactNovice</td>
</tr>
<tr>
<td>• refersTo : Participant --&gt; ProjectRole</td>
</tr>
<tr>
<td>• reportsAt : Participant --&gt; BugTracker</td>
</tr>
<tr>
<td>• simulates : FLOSSProject --&gt; LearningActivity</td>
</tr>
<tr>
<td>• takesPartIn : Participant --&gt; LearningProcess</td>
</tr>
<tr>
<td>• undertakes : Team --&gt; LearningActivity</td>
</tr>
<tr>
<td>• worksOn : Participant --&gt; FLOSSProject</td>
</tr>
<tr>
<td>• worksWith : Participant --&gt; Resource</td>
</tr>
</tbody>
</table>
3.3 Step 2: Identification and Specification of Learning Processes

With OntoLiFLOSS as a guide to study and model the identified learning processes, the next step is to essentially define these processes. This can be done through a specification language. It is a critical task of our endeavor. The idea of process definition entails specifying the activities and flow of occurrences thereof between learning actors within the settings established via the Ontology.

The current literature is endowed with extensive exploration, critiques and development of specification languages in software engineering and modeling [87-93]. These languages or methods help in simulating the behaviors and functionalities of computer programs before they are developed. The works by Overstreet et al.[94] and Cooke et al.[91] draw attention to an important role of these specification methods with regard to producing simulation models. These are models that depict a representation of some functionality as it is expected to occur in a specific domain. Furthermore, there has been an increasing interest in the area of specification languages for process modeling [95-99] and one of them, called as the Process Specification Language (PSL), provides a set of concepts and terms used for the description of process reengineering, process realization, process simulation etc.[95-96]. Another specification language that can be used for both software engineering and process modeling is Abstract State machines (ASM) as presented by Farahbod et al. [100]. Additional reports on this method are presented by Börger and Stärk [101] and more importantly Börger [102], who gives a detailed annotation of ASM biography since the inception of this area of research.

Another specification language is referred to as YAWL. YAWL is a workflow language based on a number of existing workflow patterns [117]. The need for this language stems from an evaluation conducted on several workflow products. This resulted in the detection of considerable differences in the ability to capture control flows for non-trivial workflow processes [117]. Some of these include those involving multiple instances, complex synchronizations or non-local withdrawals [117]. These patterns lack, specifically, support for the multiple instance patterns, the cancellation patterns and the generalized OR-join [118]. Therefore, YAWL was developed taking Petri nets as a starting point in order to add mechanisms allowing for a more direct and intuitive support of the workflow patterns [117].

As we set our sights on the most appropriate process specification approach to consider for our task, a number of factors are to be taken into account for this purpose. First, the most important trait is to produce workflow specifications for control-flow (or process). In essence, the approach should be able to describe tasks and their execution ordering through different constructors, which permit flow of execution control, e.g., sequence, choice, parallelism and join synchronization. More importantly, it is noted that tasks in elementary form are atomic units of work, and in compound form modularize an execution order of a set of tasks. Secondly, the process specifications produced should be in a format that can be automatically checked or verified.

Considering these two factors, we chose to represent these processes through Workflow Petri nets. Van der Aalst [121] discusses the application of Petri nets to workflow management. This is premised on the primary role of Petri nets. A number of literature studies [122-127] accentuate the role of Petri nets in modeling and analyzing processes. On the one hand, Petri nets are used as a design language for the specification of complex workflows, and on the other hand, Petri net theory provides for powerful analysis techniques which can be used to verify the correctness of workflow procedures [121].

Moreover, learning processes we are trying to model can also be described as workflows of activities (tasks) that are undertaken over a period of time through a number of states by a number of actors. At this point of our endeavor, it is critical that we provide a certain level of visualization for the processes we have identified and described so far in this work. Through these workflow Petri nets, we accomplish our critical task.
3.3.1 Workflow Petri nets for Learning Processes in FLOSS: Background and Preliminaries

In a Free/Libre Open Source Software community, the transfer of software engineering skills between participants occurs in informal fashion but can be tracked through participants’ activities. In order to accomplish this, some qualitative works on these activities have helped us identify a number of activities that make up the processes that describe how learning occurs in FLOSS [1-11, 18, 20-23, and 27]. We have a set of simple constructs through which we express the categorization and occurrence of these activities.

Firstly, in our framework three important phases that correspond to three stages of software engineering skills development cycle can be noted. These phases are Initiation, Progression and Maturation. For each stage, a number of tasks/activities are carried by both the Novice (knowledge requester) and the Expert (knowledge provider). Details of these activities in each phase or related thereof are given below. While these phases and states apply for all learning processes, the demarcation thereof can be reflected through instances of activities for each learning process. Other constructs include Participants (Novice or Expert) and Activities.

For the purpose of simplification, we ought to separate the workflows according to the three identified phases. Therefore, we provide a Workflow net for each of the two categories of participants in the learning process at different learning phases. This fosters simplicity and, above all, it makes the verification process we will carry out at a later stage as effective as possible.

3.3.2 Workflow Petri nets for Learning Processes in FLOSS: The Learning Process Nets

A learning Process is assumed to take place between members of FLOSS communities who participate in the different community forums discussions and play any role to the project. In this process, any participant can be either a knowledge receiver (Novice) or a knowledge sender (Expert) depending on the level of expertise and participant’s profile. We assume that learning occurs between FLOSS members with a diversified expertise background who learn from each other.

Therefore, it is critically important to specify here that our tentative representation of learning processes as primarily guided by the ontology and existing literature on learning in FLOSS [1-13, 18, 20-23, 27] is based on a number of considerations.

Firstly, the structure of the models with respect to sequencing or flow of activities is based on a number of factors subtly discussed in Chapter 1. These factors include role migration in FLOSS communities and the underlying need to build reputation which sustains the desire and eagerness to learn. Considering role migration, in Figure 2, Glott et al.[2] exemplifies a succession of activities as they might occur in a typical FLOSS environment. When users move from passive to active, they can only report bugs at this stage as they are simple testers or software users. However, when they move to a step higher in the hierarchy, becoming developers or co-developers, they can perform activities appropriate to this level including the ability to submit code, fix bugs or review code. This process is carried through the different roles until eventually the user can become a project initiator and perform related activities. This gives a hint regarding which activities are executed and how we can derive a flow of succession for all possible learning activities as enumerated in the studies [1-13,20-21,23,27 and 30] we explored in Chapter 1.

Moreover, Cerone [11] provides hints that can help reconstruct a sequential flow that captures learning in FLOSS communities. In describing the contribution and collaboration among FLOSS contributors,
Cerone [11] asserts that in the first phase, the purpose of interaction and communication is solely to help capture, describe and understand contents with no production activity; in the second phase, communication is used to gradually propose new contents and activities aim to defend the contents and provide feedback to other posted contents with production only at a trial and error process level; in the last phase, participants are able to get involved in project. Furthermore, Cerone [11] identifies some basic activities that are performed in FLOSS. These include observe, use, post and commit. Based on the different roles that users can play, such basic activities can be captured through a number of corresponding instance activities such as report bugs and provide feedback as part of post, submit code as part of commit etc.

The flow of activities can be justified, at least in part, following Figure 4, where Cerone et al.[23] give a representation of the contribution of individual FLOSS community members in line with the overall team work. It is demonstrated that after performing a task, individual FLOSS members will notice the need for interaction with the team, prepare for such interaction and contribute accordingly. This highlights a sense of interaction that helps us define how Novices and Experts interact.

Therefore, the nature of role migration in FLOSS and the need to build reputation which sustains the flow of activities, constitute enough ground for the consideration that a learning process in FLOSS communities is a progressive skills development process. Given these elements, we apply cognitive mapping [180-181] for synthesizing and sense making and thus, retaining 3 key constructs for the process representation:

- **Phase**: A learning process occurs in 3 distinct phases. These phases include Initiation, Progression and Maturation.
- **Basic activities**: In each of the 3 identified phases, a number of basic activities occur. These include Observation and ContactEstablishment in the Initiation Phase; Revert, Post, and Apply in the Progression Phase; and finally Analyze, Commit, Develop, Revert, and Review in the Maturation phase.
- **Instance activities**: These are variations of the basic activities as they occur in FLOSS communities. All these activities are represented in the ontology as subclasses of the LearningActivity class.

Taking into account these constructs, we build a-priori models that solely capture instance activities. Unlike in Mukala et al. [179] where basic activities can be represented as states on the ASM, the specification of the Petri nets does not allow for such notations. Therefore, we distinguish between the activities belonging to a specific set (basic activity) by annotating them with both the first letter of the participant’s identifier (Novice or Expert) and either the first letter or the first two letters of the state. For example, in the activity ContactExpert (N-O) as seen on the net in Figure 38, the two letters N and O in the parentheses refer respectively to the participant Novice and basic activity Observation.

The last consideration is that this representation of the learning processes as a-priori models [85] can also be referred to as normative models [85]. By normative models, we imply that the description provided by the models gives a comprehensive representation of how we expect the activities to normally occur given the information that we have. This information includes the basic and instance activities that are documented to occur, the context in which they occur (based on experience and level of knowledge) as well as the role migration. Currently, to our knowledge, there exists no proven framework that defines a proper sequence of learning activities in the literature.

Nevertheless, we do not claim that this representation is a conclusive depiction of how learning definitely occurs in FLOSS communities. As Greller and Drachsler [180] argue, more empirical evidence is needed to identify which pedagogic theory Learning Analytics serves best as the reliance is only on data at hand. It is rather an attempt to reconstruct the steps through which both the Novice and Expert go through
during their interactions. Consequently, our conformance analysis will help us clearly identify both the sequence of activities and their complete depiction considering the data we have at hand.

### 3.3.2.1 Phase 1: Initiation

In this phase, both Novice and Expert perform a number of activities that can trigger and perpetuate a learning process as seen in Figures 38 and 39 respectively. The basic activities include *Observation* and *ContactEstablishment*. Following the factors we discussed earlier, we can assume that in this phase, instance activities for the *Observation* basic activity occur first, followed by the instance activities of the *ContactEstablishment*.

In Figure 38, we have a representation of a possible process for the Novice in this phase. It can be distinguished 7 instance activities. Among them 5 instance activities for *Observation* as it can be seen at the end of each activity annotation with the letter O, and the last 2 are instance activities for *ContactEstablishment*. Therefore the process is projected to occur in this succession order of activities:

1. FormulateQuestion(N-O)
2. IdentifyExpert(N-O)
3. PostQuestion(N-O) or CommentPost(N-O) or PostMessage(N-O)
4. ContactExpert(N-Es)
5. SendDetailedRequest(N-Es)

Depending on the need or question from the Novice and the involvement of the Expert, a full process in this phase starts at step (1) and ends at step (5). We have three possible traces that the model offers us:

1. <FormulateQuestion(N-O),IdentifyExpert(N-O),PostQuestion(N-O),ContactExpert(N-Es),SendDetailedRequest(N-Es)>
2. <FormulateQuestion(N-O),IdentifyExpert(N-O),CommentPost(N-O),ContactExpert(N-Es),SendDetailedRequest(N-Es)>
3. <FormulateQuestion(N-O),IdentifyExpert(N-O),PostMessage(N-O),ContactExpert(N-Es),SendDetailedRequest(N-Es)>

Looking at Figure 39, the Expert performs a total 6 instance activities in this phase. The first 4 activities are part of the *Observation* basic activity, while the last 2 occur as the Expert tries to make contact with the Novice through *ContactEstablishment*. The letter E simply denotes Expert while O and Es respectively denote *Observation* and *ContactEstablishment*. We assume that these activities occur tentatively in this order:

1. ReadMessages(E-O) or ReadPost(E-O),CommentPost(E-O) or ReadSourceCode(E-O)
2. ContactNovice(E-Es)
3. CommentPost(E-Es)

This succession order of activities simply suggests that the Expert gets involved in the learning process by starting with any of the three activities in (1) and follows the sequencing until step (3). The model offers the possibility of 3 traces as follows:

1. <ReadMessages(E-O), ContactNovice (E-Es), CommentPost(E-Es)>
2. <ReadPost(E-O),CommentPost(E-O),ContactNovice(E-Es), CommentPost(E-Es)>
3. <ReadSourceCode(E-O), ContactNovice (E-Es), CommentPost(E-Es)>
Figure 38. Workflow net for Novice in Initiation phase
Figure 39. Workflow net for Expert in Initiation phase
3.3.2.2 Phase 2: Progression

In this phase, both Novice and Expert execute a series of new activities building up from the previous phase. This is shown in Figures 40 and 41 for the Novice and Expert respectively. At the heart of these models, we distinguish three basic activities: Revert, Post and Apply. Revert builds from the previous sets of activities during the Initiation phase in providing instance activities that show how both the Novice and Expert provide feedback and pave the way for the next sets of activities. With Post, the two participants perform a number of activities that revolve around posting either to the main discussion forums, source code or any appropriate platform as needed. In the last basic activity, Apply, the emphasis is on the activities that express the application of any comments, remarks or knowledge acquired as a result of the last two sets of activities. The assumption is that Revert occurs first, then Post and then Apply. Also, in the annotation, at the end of each instance activity we use the letters R, P and V to respectively denote the three basic activities.

In Figure 40, the model depicts a tentative representation of how the Novice operates in this phase. A total of 8 activities are performed by the Novice. Of these instance activities, the first 2 activities are part of Revert, the next 4 activities are instances of Post and the last 2 are instances of Apply. The following order of occurrence provides a picture of how the Novice executes these activities:

1. ProvideFeedback(N-R)
2. PostQuestions(N-R)
3. ReplyPostedQuestions(N-P)
4. PostQuestions(N-P)
5. SendFeedback(N-P)
6. AnalyzeSourceCode(N-A)
7. CommentOnCode(N-A) or ReportBugs (N-A)

The Novice is understood to follow these steps as part of the learning process and hence, 2 possible traces can be observed as follows:

1. <ProvideFeedback(N-R), PostQuestions(N-R), ReplyPostedQuestions(N-P), PostQuestions(N-P), SendFeedback(N-P), AnalyzeSourceCode(N-A), CommentOnCode(N-A), ReportBugs(N-A)>
2. <ProvideFeedback(N-R), PostQuestions(N-R), ReplyPostedQuestions(N-P), PostQuestions(N-P), SendFeedback(N-P), AnalyzeSourceCode(N-A), CommentOnCode(N-A), ReportBugs(N-A)>

After accepting a request from the Novice, the Expert performs a number of activities contributing to the learning process as seen in Figure 41. The Expert thus performs a total of 12 instance activities in this possible order of execution:

1. ReviewThreadPosts(E-R) or ReviewThreadCode(E-R)
2. SendReply(E-R)
3. PostQuestions(E-P) or ReplyPostedQuestion(E-P)
4. SendFeedback(E-P)
5. ReportBugs(E-P)
6. ReplyToPost(E-A)
7. RunSourceCode(E-A) or AnalyseSourceCode(E-A)
8. ReportBugs(E-A) or CommentOnCode(E-A)

This sequence of activities gives us an indication of possible traces of activities that the Expert can leave in this phase. 16 traces are possible from this model, 4 of which are given as follows:

1. <ReviewThreadPosts(E-R), SendReply(E-R), PostQuestions(E-P), SendFeedback(E-P), ReportBugs(E-P), ReplyToPost(E-A), RunSourceCode(E-A), ReportBugs(E-A), CommentOnCode(E-A), ReportBugs(E-A)>
2. \(<\text{ReviewThreadPosts}(E-R), \text{SendReply}(E-R), \text{ReplyPostedQuestion}(E-P), \text{SendFeedback}(E-P), \text{ReportBugs}(E-P), \text{ReplyToPost}(E-A), \text{RunSourceCode}(E-A), \text{ReportBugs}(E-A)\)>

3. \(<\text{ReviewThreadCode}(E-R), \text{SendReply}(E-R), \text{ReplyPostedQuestion}(E-P), \text{SendFeedback}(E-P), \text{ReportBugs}(E-P), \text{ReplyToPost}(E-A), \text{RunSourceCode}(E-A), \text{CommentOnCode}(E-A)\)>

4. \(<\text{ReviewThreadCode}(E-R), \text{SendReply}(E-R), \text{ReplyPostedQuestion}(E-P), \text{SendFeedback}(E-P), \text{ReportBugs}(E-P), \text{ReplyToPost}(E-A), \text{AnalyseSourceCode}(E-A), \text{ReportBugs}(E-A)\)>

…
Figure 40. Workflow net for Novice in Progression phase
Figure 41. Workflow net for Expert in Progression phase
3.3.2.3 Phase 3: Maturation

This last phase of the learning process asserts how the Novice has mastered the skills learnt during the learning process. During this phase, the learning process intensifies as the knowledge exchanged has matured and allows for the Novice to perform more advanced activities according to the progress made in skills accumulation, as depicted in Figure 3.

The Novice is assumed to have acquired enough skills to be able to undertake a set of activities ranging from reviewing posts, reviewing code to fixing bugs, analyzing committed code to identify bugs and producing code while the Expert plays more of a monitoring role. At this stage, there are 5 basic activities that the Novice performs. These include Analyze, Commit, Develop, Revert and Review. The letters A, C, D, R and Rv respectively annotate instance activities relating to the basic activities as executed by both the Novice and Expert this phase.

In Figure 42, the Novice performs 18 instance activities in an arbitrary order as follows:

1. AnalyzeDiscussions(N-A), AnalyzeThreadProgression(N-A) or AnalyzeSourceCode(N-A)
2. SubmitBugReport(N-C) or SubmitCode(N-C) or SubmitDocumentation(N-C)
3. FixBugs(N-D) or GiveSuggestion(N-D) or PostCommentOnCode(N-D) or ReplyToSuggestion(N-D) or WriteSourceCode(N-D) or ModifySourceCode(N-D)
4. ProvideFeedback(N-R) or PostQuestions(N-R)
5. ReviewSourceCode(N-Rv) or ReviewPosts(N-Rv)
6. ReportBugs(N-Rv) or ReviewCommentContents(N-Rv)

The Novice can possibly produce numerous unique traces considering the combination of instance activities from step (1) to step (6). 3 possible traces include:

1. <AnalyzeSourceCode(N-A), SubmitBugReport(N-C), FixBugs(N-D), ProvideFeedback(N-R), ReviewSourceCode(N-Rv), ReportBugs(N-Rv)>
2. <AnalyzeDiscussions(N-A), AnalyzeThreadProgression(N-A), SubmitBugReport(N-C), FixBugs(N-D), ProvideFeedback(N-R), ReviewSourceCode(N-Rv), ReportBugs(N-Rv)>
3. <AnalyzeDiscussions(N-A), AnalyzeThreadProgression(N-A), SubmitBugReport(N-C), FixBugs(N-D), ProvideFeedback(N-R), ReviewSourceCode(N-Rv), ReviewCommentContents(N-Rv)>

The Expert will perform exactly the same activities while tracking the Novice’s progress as seen in Figure 43. The following steps capture how the Expert is expected to carry out the 19 activities as part of the learning process:

1. AnalyzeSourceCode(E-A) or AnalyzeDiscussions(E-A), AnalyzeThreadProgression(E-A)
2. SendFeedback(E-A)
3. ReviewDocumentation(E-C) or ReviewCode(E-C) or ReviewReport(E-C)
4. SendFeedback(E-C)
5. RunSourceCode(E-D), ReportBugs(E-D) or AnalyzeSourceCode(E-D), CommentOnCode(E-D)
6. ReviewThreadCode(E-R) or ReviewThreadPosts(E-R)
7. SendReply(E-R)
8. ReviewSourceCode(E-Rv), ReportBugs(E-Rv) or ReviewPost(E-Rv), ProvideFeedback(E-Rv)

The execution of these activities culminates into a number of unique traces, among which the following 3 possible traces:

1. <AnalyzeSourceCode(E-A), SendFeedback(E-A), ReviewDocumentation(E-C), SendFeedback(E-C), RunSourceCode(E-D), ReportBugs(E-D), ReviewThreadCode(E-R), ReviewSourceCode(E-Rv), ReportBugs(E-Rv)>

Process Models for Learning patterns in FLOSS repositories
In Chapter 9, as we perform a conformance verification of these traces for all 3 phases, we semantically look at the differences between these processes (traces) and the actual behavior we observe from the Event Logs.
Figure 42. Workflow net for Novice in Maturation phase
Figure 43. Workflow net for Expert in Maturation phase
3.4 Conclusion
In this chapter, we laid another foundation to guide the process discovery and facilitate conformance analysis for validation of discovered processes. We accomplished two main tasks by defining the language (OntoLiFLOSS) for terms needed to describe learning processes in FLOSS and providing a visual representation of learning behavior in FLOSS (a-priori models) based on literature studies [1-13,20-23,27 and 30].

The 3 learning phases provide a broad and generic representation of how communication between FLOSS members also involves learning processes. Such description is critical to the understanding of learning processes in FLOSS environments as it lays the details pertaining to activities and tasks performed by both the Novice and Expert as part of these processes. As pointed out, our aim is to further explore these learning processes by providing empirical evidence directly from data. The key to such empirical analysis is to make sure that we present a systematic way of mining the FLOSS repositories we have identified and producing the evidence of learning processes in these repositories. Such evidence, in the form of traces accounting for users’ activities in FLOSS communities, can be represented through process maps.

This is logically the last task in tracing learning processes. The idea is to demonstrate and express how the new acquired skills have impacted and contributed to the learner’s activities through his/her overall involvement in the project.

As these activities in different phases unfold and are completed, it is critical to note that the representation of any learning process is enriched with an analysis and description of the impact of newly acquired skills on the Novice’s contributions pre- and post- learning on a given project. And naturally, we can see this translates into contributions that the learners make over the course of the process.

In closing, we hope that these Workflow Petri nets are enough, at this point at least, to draw a representational idea of the degree to which our identified learning processes occur in FLOSS. In the subsequent chapters, we explore and analyze learning processes extracted from FLOSS data.
4.1. Introduction

Process models, as process discovery outputs, are graphical depictions of executed activities found in audit trails and log files from any information processing system. They constitute a form of evidential traces to account for users’ activities while performing their tasks in any environment that stores this information. Figure 44 depicts the Meta Data Model illustrating the basic building blocks of Process Mining.

![Process Mining Meta Data Model](image)

The idea as expressed in the model is that a log, called an Event Log, that is ready for Process Mining should abide by a number of structural properties to facilitate its processing and analysis thereof by the existing tools. Simply put, an Event Log should contain data organized and clustered in processes, each of these processes have instances uniquely identifiable with a set of activities. A process instance can also be referred to as a case instance and can include a number of events that consist of activities being executed at a given point in time.

An example could be a log for an insurance company containing information about a billing and refund process. A refund process has a number of process instances uniquely identified by the claim number. Activities that should be executed in the refund process may include registering the claim, and checking the insurance policy. An example of an event is “On Thursday September 23, 2010 Alice checks the insurance policy of the persons involved in claim 478-12”. Given such information, the goal of Process Mining can be, through its techniques, to derive abstract graphical representations of the process control flow, detect relations between the individuals involved in the process and their tasks, and infer data dependencies between different process activities.
In light of this, we consider these graphical representations as a viable means of visualizing traces of learning processes activities as they chronologically occur in FLOSS repositories. Making use of data recorded from FLOSS projects, Process Mining can help reproduce representation sketches about the inherent activities sequencing in the form of process models. However, to our knowledge, there have been very limited or no attempts at all in Process Mining FLOSS repositories for any form of analysis or investigation. As hinted in Chapter 1, a number of challenges or constraints could explain this. Some of these constraints include perhaps the structure of data records in FLOSS repositories and the non-discovery of the potentials of Process Mining. We believe this work can make an invaluable contribution in this regard in spearheading the consideration of Process Mining techniques for the analysis and depth mining of FLOSS data to answer the many arising questions in these communities as suited.

Therefore, in this chapter the main objectives are twofold. First, we try to contextualize the use of Process Mining and its relevance in light of our research endeavor. Lastly, we succinctly discuss the preliminaries and technical prerequisites for the application of Process Mining techniques to our data sets before we conduct our experiments.

### 4.2 Process Models for learning process: Crucial landmarks

In our context, the primary task in Process Mining FLOSS repositories is to construct and generate an Event Log. Given the dynamism and multiplicity of participants’ activities and behavior in FLOSS communities, it is a challenge to have a unified Meta Data Model [150-152]. At this point, literature about FLOSS data suggests that FLOSS data records files do not adhere to the basic structure as described in the Process Mining Meta Data Model in Figure 44. Understandably, a number of factors can be considered to explain this discrepancy. Some of these include the disparity and sometimes the incompleteness of data or its lack thereof making the identification and specification of cases, process instances or events in these repositories tedious.

However, all these challenges can be overcome depending on the purpose of Process Mining with regard to the type of analysis one wishes to conduct and the goals to be reached. As long as the objectives are clear, a number of attempts can be made to restructure and format the data sets in FLOSS. Our chief goal is to identify learning activities that make up learning processes, the participants involved in these activities as well as the chronological occurrence of these activities and artifacts involved. Based on this information, we decide on cases, formulate process instances and produce process models accordingly. Therefore, we present a number of guidelines that can help in constructing Process Models from FLOSS data:

- A process can be traced by considering commits made by a contributor for a given period of time where each commit could be a case id and all activities pertaining to this commit would constitute a process. The point will be to identify a learning experience through <<Role-Playing>>. This means, on the assumption that a Novice learns from other contributors’ criticism and reviews. Hence, the Novice would be the committer and Reviewers would be Experts.

- Alternatively, a process could group email messages about a topic or comments in a discussion forum about a specific question for a given period of time. Our analysis could detect and identify activities we can cluster as learning activities as well as the actors involved. It can show that for a number of activities, some users have been Novices while they have been Experts through another set of activities.

- A FLOSS project, can be a process where we cluster, all activities that have taken place since its inception and try to group them into <<Novice-Activities>> and <<Expert-Activities>>.

- Even further, commits about a component or a bug is material for a process. In this case, the component would be a case id and as with previous propositions, the goal is to cluster the traces of activities and graphically display them through a process model.
All of these assumptions can be verified and evaluated accordingly with data. More precisely and within the confine of our research objectives, these assumptions provide a way to construct process models and identify at which phase of the learning process identified activities can be assimilated to.

### 4.2.1 Heuristics

As explained in Chapter 2, an extensive review of current approaches to mining data in FLOSS is based on some initial assumptions of which some have been verified and some not really verified. A considerable number of these approaches have demonstrated that mining software repositories is achieved almost entirely based on a consideration of heuristics. In our case, the Workflow net models described in the previous chapter detail the specifications for a complete learning process that can be traced in the event of data availability. Given the nature of FLOSS data available today, we capture as much as possible the traceability of the learning activities and validate the models as needed. Nevertheless, our approach should be able to capture and explain the patterns as needed based on a number of heuristics as summarized in the table below. A number of assumptions based on the literature and heuristics will guide the generation of Event Logs. Specifically, with no automatic analysis of FLOSS repositories and given the availability of data, the logs should reflect the following activities as summarized in Table 4.

<table>
<thead>
<tr>
<th>LEARNING PHASES</th>
<th>REPOSITORY</th>
<th>TASKS</th>
<th>EVENT CASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiation</td>
<td>Messages</td>
<td>Observe/Make Contact : Novice (read, comment…), Expert (read, comment, post…)</td>
<td>Message Title, Discussion Topic, Artifact Thread</td>
</tr>
<tr>
<td></td>
<td>(Mailing/Forums/CVS comments)</td>
<td>Reply/Post : Novice(send, reply, post), Expert(reply, report).</td>
<td>Commit, Message Title, Discussion Topic, Artifact Thread,</td>
</tr>
<tr>
<td></td>
<td>Messages, Bug Tickets, CVS/SVN</td>
<td>Apply : Novice (run, analyze, comment…), Expert (run, analyze, comment…)</td>
<td></td>
</tr>
<tr>
<td>Maturation</td>
<td>MailingLists/Forums/CVS comments/ Internet Relays Messages, Bug Tickets, CVS/SVN</td>
<td>Analyze/Review/Revert : Novice (review, modify, submit…), Expert (Revert, analyze, guide…).</td>
<td>Message Title, Discussion Topic, Artifact Thread, Internet Relays Messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Develop/Commit : Novice(modify, report, develop), Expert(run, analyze, report)</td>
<td></td>
</tr>
</tbody>
</table>

Table 4 summarizes in a general way some assumptions on the types of activities we aim to trace, where they can possibly be retrieved and the possible candidate for event case as will be expressed in the process models. The idea is that activities expressing the initial phase of learning can be traced by looking at Messages and remarks in MailingList, Forums and CVS and at this stage, the type of activities to be traced will be observing and establishing contact through reading and commenting for both participants. And in this instance, either a title in a discussion forum and MailingLists or an artifact thread can be taken.
as a process. The same applies for the remaining two stages to be considered mutatis mutandis. The column task identifies arbitrarily the possible type of activities we can terrace as they pertain to learning processes.

### 4.2.2 Mining data sources

We have established, at least with regards to our objectives, the context in which Process Mining is possible for our analysis. We considered the kind of data elements and information we need to get from FLOSS repositories in order to build Event Logs. However, we still need to consider how this data is to be obtained. It is evidently clear that FLOSS repositories do not store plain-text versions of activities. This implies that we cannot specifically find activities, clearly stated (i.e PostComment, PostQuestion) in any of the FLOSS repositories including CVS, SVN, Mailings or even forums.

Therefore, we argue that given the many interests and research opportunities arising in FLOSS, finding the right data involves an important process altogether depending on the research questions. Each question sparks the type of mining techniques that are suitable to unearth the relevant data. A lot of previous approaches for mining FLOSS repositories are so far limited to exploring users’ activities and interactions from a statistical point of view. We reviewed a number of techniques and approaches in this regard in Chapter 2. Nevertheless, the objectives of our work require a semantic analysis of message and activities contents. Therefore, we consider text mining. This can help in looking at text categorization and clustering: explore emails, identify contents and cluster them according to learning activities, phase and originator.

After analysing a number of available text mining tools, a shortlist was retained to study to what extent they could aid in classifying and identifying data in our work. Some of these included Carrot2, GATE, OpenLP, RapidMiner and KH Coder. After exploring and study them in light of the data available for our study, the following remarks were drawn:

- **Carrot2**: Open source tool, but works on unstructured texts, especially webpages only. Therefore, it is not helpful to the kind of analysis we wanted to conduct such as mining email messages.
- **GATE**: With extensive documentation about its use, this is a tool whose main purpose is to annotate documents. The GUI, called GATE Developer is quite user-friendly and allows seeing a loaded document as well as annotations on the documents once created. It includes a number of plugins called CREOLE plugins as well as an information extraction system called ANNIE. These also produce an annotated document or web page… With no ability to predict or deduce activities from a text, this tool was also useless for our endeavor.
- **OpenLP**: This series of tools provide codes for sentences detectors, tokenizes, parsers etc. that can be embedded in java plugins but also not directly helpful.
- **Other commercial tools** also considered for this purpose included ClearForest Text Analytics, Omniviz, Inxight Smart Discovery® Extraction Server, Categorizer XeLDA ReXtraction Terminology, Suite™ Skill Cartridge™ library, VantagePoint, Thomson Data Analyzer, and Integrator & Clustering Engine. None of these tools embodied the capabilities required for the type of extraction analysis needed for our work.

A complete review of some of these tools and more can be found in [153-157]. In addition to looking at potential text and data mining tools, we considered other previous “famous” text analysis projects such as the ENRO email analysis done by UC Berkeley Enron Email Analysis group [157]. The general remark after looking at this approach is that it is about annotations and the adoption of such techniques would not yield the kind of output we need to create process models.
Since all of these tools and approaches work mainly on unstructured documents, text files and webpage and do not provide capabilities for automatic prediction or deduction of activities, we considered a more suitable alternative using Semantic SQL introduced with Microsoft Server 2012.

4.2.3 Semantic Search with SQL

This is a novel method that comes with Microsoft SQL Server 2012 and is based on three linear scale, incremental and fully automatic semantic mining algorithms as explained in [158-162]. These three algorithms give rise to three weighted physical indexes: Tag Index (top keywords in each document); Document Similarity Index (top closely related documents given any document); and Semantic Phrase Similarity Index (top semantically related phrases, given any phrase), which are then query-able through the SQL interface [158].

While general term similarity and/or document similarity are computed via compute intensive algorithms, such as, for example, Latent Semantic Indexing (“LSI”) and Latent Direchlet Allocation (“LDA”), this method extends on these algorithms by deriving document similarity indices [158].

In essence, this approach derives a document similarity index for a plurality of documents. The same can be applied for a body of text. In our data sets, we look at entire email messages that can be queried in SQL SERVER applying these techniques.

The process is such that that a document or body text is accessed then a tag index including one or more keyword/weight pairs is computed for the document. Each keyword/weight pair maps a keyword to a corresponding weight for the keyword to indicate the significance of the keyword within the document. A specified number of the most significant keywords in the document are identified based on weights in the tag index [158-161]. Given the context of this thesis, we think it is out of scope to detail the technicalities and low-level specifications of this method. Therefore, we refer the reader to the work of Mukerjee et al. in [158-162] for more an indepth discussion of these algorithms and methods.

Nevertheless, from a practical perspective, the semantic analysis performed by SQL Server is essentially a way to improve search accuracy by providing capabilities that provide results by understanding the intent from the user as well as based on contextual meaning of terms through a number of phases as seen in Figure 45. By providing a set of key phrases or keywords, the algorithms can consider the semantic and contextual meaning of these input data and retrieve the document or body of text that fits the search criteria. Unlike a normal match-based query, Semantic Search allows the user to search the data corpora based not on terms but on their meanings, their semantics.

![Figure 45. Phases of Semantic Indexing](image-url)
Executing Semantic Search is premised on a wide range of technical prerequisites. Such preconditions include installing the Full-Text and Semantic Extractions for Search feature, installing the Microsoft Office 2010 Filter Packs and Service Pack 1, installing, attaching and registering the semantic language database, creating a full-text catalog and creating a full-text index with the Statistical_Semantics option enabled.

Furthermore, a number of steps are required in order to configure the environment and enable writing T-SQL queries that specify semantic search. In order to activate Semantic Search on our repository, we are required to ensure that full-text search is enabled first. Full-text search in SQL server 2012 allows approximate searches in databases. Another requirement is that before using full-text predicates and functions, it is required to create full-text indexes inside full-text catalogs. An example of a statement that installs the Semantic database and attaches is as follows:

```
CREATE DATABASE semanticsdb
    ON (FILENAME = 'C:\Program Files\Microsoft Semantic Language Database\semanticsdb.mdf')
    LOG ON (FILENAME = 'C:\Program Files\Microsoft Semantic Language Database\semanticsdb_log.ldf')
    FOR ATTACH;
GO
```

In the next chapter, we detailed on some of the key queries written in order to configure our data sets and follow all the necessary steps to enable semantic querying. At this point though, we decide on how to index our repositories for semantic search and what potential attribute needs to be specifically configured to enable the search. We proposed earlier that given the structure of the available data, our logs shall be based on redefined key factors including the type of information (attributes from tables) and role of the users. The idea is to focus on key attributes where a number of identifiable learning activities can be traced from as well as the role that each user plays during this process. While in the next chapter a more detailed description of our data set is given, we introduce here as part of preliminaries the key attributes per table and database that require semantic search. These include:

<table>
<thead>
<tr>
<th>REPOSITORY</th>
<th>ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mailing_lists</td>
<td>Message (subject,body)</td>
</tr>
<tr>
<td>Forum (IRC)</td>
<td>message</td>
</tr>
<tr>
<td>CVS (Code reviews)</td>
<td>Text from comments and issues</td>
</tr>
<tr>
<td>Source_code</td>
<td>message</td>
</tr>
<tr>
<td>Bug Tracking</td>
<td>Text from comments, summary and description from issues</td>
</tr>
</tbody>
</table>

With ingredients such as the mining technique, the heuristics and the potential attributes for our Event Logs in place, the next step lies on constructing the actual Event Logs.

### 4.3 Constructing Event Logs

In light of all these preliminaries discussed above, we shall construct the logs to be processed in order to generate our process models according to the phases. Although we have identified all key important factors and elements that lead to this stage, we cannot analyze our data sets with Semantic Search without the keywords and key phrases that determine the learning activities during the process for the initiation, progression and maturation phase. As a prerequisite for semantic search, in this section we describe our methodology in generating the key phrases needed for the analysis.
Chapter 4: Process Models for Learning Processes: Preliminaries

Therefore, constructing the Event Logs depends on 2 important factors: specifying the potential keywords for each activity from this phase as well as Semantic Search based on these keywords.

4.3.1 Generation of key phrases

In the attempt to find the best way to generate key phrases that can identify and classify learning activities from FLOSS repositories, we briefly describe two options that could possibly be considered. The first option is try to find a way of automatically generating and retrieving these terms or group of terms from the text to be mined. A number of studies and works in this direction have been conducted and were quite successful in their own right. However, this success has proved insufficient to consider for our purpose. The second option, consisting of works from literature on previous studies in FLOSS, offers a more suitable alternative.

4.3.1.1 Automatic assignment/discovery of key phrases

With this option, key phrases are words or short phrases, are taken in the context where they are commonly used by authors of academic papers in order to characterize and summarize the topics covered by a document [104]. Most often, noun phrases between certain ranges depending on the specifications for a paper will determine the length of such key phrases. Examples include digital libraries, browsing interfaces, text mining, keyphrase extraction, and machine learning. One can also consider the index terms collected in “back-of-the-book” subject indexes [104].

Gutwin et al. [104] present an approach that aims to search documents based on the availability of such key phrases. This implies that the papers or any documents index with key phrases are saved in some repository and a user can search using a certain keyword or key phrase and the corresponding document will be displayed. Key phrases are used as basis for a searchable index, since they represent the contents and subject matter of a document [104].

However, sometimes and in some document collections, author-specified key phrases are not available for the documents and this is where automatic generation of key phrases is required. The methodology proposed by Gutwin et al. [104] also assume that automatic determination of key phrases works in one of two ways: meaningful phrases can either be extracted from the text of the document itself, or may be assigned from some other source (but again using the document text as input). This will nowhere help us classify messages in FLOSS repositories and produce learning activities. The methods certainly are useful and can be used for other purposes even on FLOSS data but cannot help for our type of analysis.

A lot of work done in this regards for decades has been about finding key phrases as specified in published papers and in return match these with the queries to locate these papers. Equally, the current literature on key phrases and text mining appears to focus more on the same approach for key phrase matching for documents search. This means that there is a set of key phrases that will be matched to whatever the user inserts and the query returns data that have been indexed to these key phrases. In such scenarios, a user would insert a key phrase and the system in turn would return and retrieve all key phrases that match the inserted and the corresponding documents.

Some drawbacks in trying to generate missing indexes are that a certain number of automatically identified index terms inevitably contain errors that look downright foolish to human eyes [105]. Indexes consisting of automatically identified terms have been criticized on the grounds that they constitute indiscriminate lists, rather than synthesized and structured representation of content. And because computer systems do not understand the terms they extract, they cannot record terms with the consistency expected of indexes created by human beings [105]. Nevertheless, the emphasis has remained that index
terms can be generated in a library or corpus of documents so that when there is a query or search, the input phrases are paired with identified index terms that are linked to documents which will be displayed.

Another attempt in working with key phrases and indexes is presented by Sahami et al.[106]. In this study, a system called Service for Organizing Networked information Autonomously (SONIA) is introduced. It is a system for topical information space navigation that combines query-based and taxonomic systems approaches. Using machine learning techniques, the system executes a sort of dynamic document categorizations based on the full-text of articles that are retrieved in response to users’ queries [106]. Furthermore, SONIA employs a combination of technologies that takes the results of queries to networked information sources and, in real-time, automatically retrieve, parse and organize these documents into coherent categories for presentation to the user [106].

Pierre [107] does the same thing from a different angle. His work is motivated by the assumption that vast amount of content is essential for realizing the web’s full potential as an information resource. In order to exploit this information resource, Pierre believes that a consistent use of metadata and other descriptive data structures such as semantic linking is required. Therefore, he advocates categorization as an important ingredient in searching the web. Examples of web directories that exploit categorization include Yahoo!, Looksmart, and Open Directory Project etc. [107]. Although advocating categorization, it is not about automatic discovery of key phrases but rather classification of web content based on their key words that can, once again, be queried in order to display them.

Lastly, we consider an important factor that constrains the automatic generation of key phrases specifically in FLOSS communities. Boldyreff et al. [108] conducted an extensive study and reported that the language used in FLOSS communities cannot be predetermined. The common understanding is that in these environments, communication channels are “volatile”. These channels include IRC, mailing lists and forums. Design over such channels is difficult because of its iterative and visual nature and the textual nature of the communication protocols [108].

Reasons differ from platforms to platforms and projects. The study argues that GNOME was dubious as to how useful mailing list and IRC discussions could be. Also, they thought that these communication methods, in addition to a complex bug database, may intimidate non-technical users (such as usability engineers). This can be especially true when developers begin discussing technical details and implementation. Mozilla experienced community and consent problems with having a too-open (open-to-public) or too-closed (invite-only) mailing lists. NetBeans reported the problem of fragmented discussions between bug reports and mailing lists, which can also happen across different mailing lists. OpenOffice made a recommendation of setting up a “collaborative, visual pace” to help developers and designers more easily communicate visual and interactive ideas [108].

While these are crucial reasons that can help foster communications in FLOSS communities, our focus however lies on analyzing the actual recorded data. All these existing approaches to key phrases-based querying, do not meet our requirement because they do not execute what the purpose of our analysis is. The context in our endeavor is not to find matching key phrases but rather to determine with the help of key phrases the type of activities undertaken when an email is sent, or a comment is left on source code, or when people have exchanged messages during chats. These activities, labeled learning activities constitute the building blocks of the process maps. Therefore, the key phrases and rules in this regard cannot be determined automatically. Nevertheless, they cannot just be decided arbitrary.

Hence, we consider the second alternative that is based on previous studies and analyses in FLOSS communities to understand how people work together, how they communicate and what type of key phrases can be found as a result.
4.3.1.2 Previous studies on users’ interactions in FLOSS communities

The first study is carried out by Singh et al. [109]. The authors discuss ways in which help and interactions are conducted in FLOSS communities, specifically technical support. They explore online technical support of open source software by a study of postings to discussion boards. The study results indicate that there are several types of details that are required by the help-givers to be able to diagnose and remediate help-seekers’ difficulties [109]. It can also be argued that the environment typically includes discussion forums or mailing lists to which users can post questions and get help from developers or other users [109]. These forums are unrestricted and act as a learning environment for Novices and Experts alike.

Furthermore, succinctly the objective in this work has been on understanding the way discussions occur, understanding the nature of the discussions in these forums with the hope that an examination of the more complex and problematic help-giving interactions may lead to finding ways in which those interactions might be improved. Also, a key motivation for this study was the lack of consideration of end users and their problem solving approaches. How do people get help and how do they seek this help in FLOSS communities? This is the question at the heart of the investigation [109].

Moreover, Lakhani and von Hippel [110] provide some insights on the technical help for OSS, but it is mostly concerned with motivation, trying to answer why people bother to help others, and not really examining how the help unfolds, which is our main interest. Although this paper has generated a lot of citations, it appears that most of these papers or even all of them are mainly concerned with the motivation problem rather than the process of help-giving.

On the other hand, Singh et al. [109] argue that this lack of interest in focusing on technical help or help giving process may be that technical help is considered either trivial, or just not interesting enough to merit attention, other than amazement that anyone who wasn't being paid would bother to do it. Even so, Lakhani and von Hippel [110] call technical help a “mundane but necessary task”. Very necessary indeed as it helps shed more lights on how interactions occur in FLOSS environments.

Unlike utilities like FAQs, or telephone lines for commercial software where people would call to request help ensuing an interaction with a series of questions, OSS technical support is provided by volunteers who are themselves users of the software and some of whom may also be involved in the development of the software, providing an invaluable design feedback loop. Even the most enthusiastic help-giver is only going to spend a small fraction of their time on help-giving, and although frequently having substantial technical Expertise, may not be skilled in the best ways to give help [109].

This observation may not have a direct impact on how useful a feedback could be, but for our objective, the intention is to determine the process of feedback-giving whatever fit it seems for FLOSS. It is evidently shown that things occur in a non-orthodox fashion than they would in traditional software user or developer’s settings, and this is an accepted FACT. However, it is not clear from literature how this happens in FLOSS. It is important to understand this process of help-giving and the settings of such discussions as it this allows help-seekers to phrase requests in terms of the environments and discussions context work goals, rather than purely in terms of an application's functionality [109].

In the attempt to alleviate this problem, a number of OSS projects were investigated in order to analyze the interactions pertaining to help-giving and thus, understand how this occurs and key phrases involved [109]. Furthermore, Singh et al. [109] chose 10 problems from 5 FLOSS platforms and 15 problems from
2 others that happened to have particularly rich sets of data. Focusing on topics and thread with multiple responses as summarized in Figure 46.

<table>
<thead>
<tr>
<th>Project</th>
<th>Number</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opera</td>
<td>15</td>
<td><a href="http://www.opera.com">www.opera.com</a></td>
</tr>
<tr>
<td>Greenstone</td>
<td>15</td>
<td><a href="http://www.greenstone.org">www.greenstone.org</a></td>
</tr>
<tr>
<td>Gnome</td>
<td>10</td>
<td><a href="http://www.gnome.org">www.gnome.org</a></td>
</tr>
<tr>
<td>Mandrake</td>
<td>10</td>
<td><a href="http://www.mandrakeuser.org">www.mandrakeuser.org</a></td>
</tr>
<tr>
<td>KDE</td>
<td>10</td>
<td><a href="http://www.kde.org">www.kde.org</a></td>
</tr>
<tr>
<td>BCS</td>
<td>10</td>
<td><a href="http://www.bcs.org">www.bcs.org</a></td>
</tr>
<tr>
<td>Mozilla</td>
<td>10</td>
<td><a href="http://www.mozilla.org">www.mozilla.org</a></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>80</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 46. List of FLOSS projects used to study the process of help-giving among users [109]**

In order to analyze the data collected, authors made use of Grounded Theory as a basis for data analysis, an approach that has been used in other OSS and online content analyses [109]. The choice of this methodology is based on evidence that in the process of grounded theory development, theory generation and development is done inductively by studying the phenomenon it represents. Concepts are discovered, developed, and provisionally verified through systematic data collection and analysis. One does not begin with a theory, and then prove it. Rather, one begins with an area of study and what is relevant to that area is allowed to emerge.

They, thus, studied all the interactions in these projects and developed some basic concepts, producing 30 categories of key phrases used during the course of interactions among FLOSS users. These were then further grouped into five broader categories. Given the small size of the study and its limited focus noted above, our findings are tentative. Figure 47 depicts these categories. We note the categories, provide an example from the data set and discuss them.

While very detailed descriptions of these categories by groups have been given in [109,114-115], we cannot reproduce such level of details in this thesis and therefore refer to the original papers for ample details. Nevertheless, the five types of questions asked are of direct high relevance to our work and therefore we briefly describe some of these and provide examples.

The first type of questions that can be posed in FLOSS communities and to which Experts can respond is “How to” questions [109]. These are the kind of questions asked by users on either discussion forums or mailing lists when they did not know how to use a particular feature of the software they were using or when they were not able to complete a task that they were trying to accomplish [109]. One can directly identify a message containing such questions as coming from a Novice. In general, these “How To” questions are simply formulated. An example of such questions include How do i put my old MIE bookmarks into Firefox?[109], How can I fix this error?
In the second category of questions, authors found out that most help-seekers would ask “What is wrong” questions [109]. Almost self-explanatory, these questions were asked when the help-seeker thought that she was doing everything right and still could not get her work done. The user might either think that she knows what she wants to do and how to do it but with no success or she can be clueless about what the problem is or what steps are required to solve a typical problem. The feeling in terms of these questions is that comparatively more details were provided and they had a format in which the help-seeker said what she was doing, what she did and then what happened [109].

**Example**
I use SuSE 9.1 and upgraded to kde 3.3, and when I login the system tray says that it is empty, but when I remove it and put it back it is full of stuff like it is supposed to. I tried renaming ~/.kde to ~/.kde− and it still didn't work. What is wrong? [109]

The last category of questions we describe here is the “I am stuck” kind of questions. These are the kind of questions where the help-seeker is unable to go ahead or make progression and needs urgent help. This urgent help entails external intervention from FLOSS communities’ members. In some cases, the help-seeker could not proceed with his troubleshooting skills and was at a loss as to why his system is not doing what he wanted it to do [109].
Chapter 4: Process Models for Learning Processes: Preliminaries

Example

Hi, I am running KDE 3.2.3–1.0.1 (from kde-redhat) on Fedora Core 1. One user has recently complained of being unable to login. When he enters his username and password into KDM, the screen goes blank briefly and then the login screen reappears. He could log on earlier in the day, and claims to have changed nothing. Other users can still logon. I have checked that he has a proper shell (/bin/bash), and tried deleting ~/.kde, /tmp/orbit~ , /tmp/mcop~ , /tmp/ksocket~ , /tmp/kde~ , /tmp/gconfd~ and /var/tmp/kdecache~. What else could be the problem, and how can I fix it? [109]

All these questions eventually generate responses that were also studied to understand how the feedback processes worked. More importantly, the most frequent key phrases are identified and described. However, before answers are given, sometimes the help-givers would request more details. Some of these response key phrases can be noticed by considering the category as shown in Figure 47. These are posted by help-givers when the information provided by the help-seeker was not complete enough to judge what the problem was. The diagnosis or solution is dependent on the type of problem or even the system that the help-seeker was using. In such case, help-givers would ask for certain details in order to clarify and understand the extent of the problem before assisting.

Example

Hello, We can almost certainly help you with the problem, but not with the amount of information you have given us about it. Useful information you could give us would include: [109]

1. the operating system you are using (Windows Me/ Windows 95/ Linux/ MacOs/ etc)
2. some indication of the nature of the data you're trying to import including (a) the file format (doc/pdf/ps/bib/html/avi/etc) (b) the language of the text in the documents (English/ German/ Chinese/ etc) and (c) whether the data is in a few large files or many small files
3. the version of the (software name) you are using
4. the command line you are using
5. the last line that appears on the screen before it hangs

Alternatively, it was noticed that in some instances the question came in a more terse form like “What are the details of your system?” which can lead to incomplete details, yet more iterations and hence longer time in problem solving [109]. The operating systems and software details are needed in most of the cases to understand what is going on.

Furthermore, the help-givers would require a bit more details pertaining to understanding the history before the problem was encountered by the help-seekers. Hence, questions asked by the help-givers attempt to understand the activities or actions the help-seeker had taken that led her to the problem and provide some details about these actions [109].

Example

Some more details would help. When you say “About 25%”, what does that mean in real terms? One screenful, perhaps? One paragraph?
Next question, how are you pasting it? Did you select Copy in OoWriter and then Paste in Opera, or are you using the Linux middle-click paste feature?
Third question - system details. Version of Linux, desktop that you use, etc. I know that KDE can be somewhat strange about pasting from one app to another, usually clicking on the Klipper icon (that little clipboard on KDE's task manager) and selecting the text to paste generally works.”

Apart from these follow-up questions, help-givers can provide answers directly to the questions depending on the complexity of the questions themselves and the level of details. In some instances, the
answer could be a step by step guide as specified in Figure 47 or simply provide a list of possible solutions.

**Example**

Hi, I can think of a couple of things to try:

1. [http://gsdl/collect/science/import/page.html](http://gsdl/collect/science/import/page.html) this won't work as it is looking for an internet server named “gsdl” and then /collect on that server. You could try/gsdl/collect/science/import/page.html but I don't think it's a good idea to link to the import directory. Greenstone can handle internal links...

2. If you really want to give hard-coded links, edit your collect.cfg file so that for HTMLPlug you include a certain option, like: plugin HTMLPlug -nolinks This means that greenstone won't do any interpretation of the links and they will be displayed exactly as they are in the source documents.

3. You could use the “-file_is_url” option to HTMLPlug as above. This is normally used when building a collection from a web mirror, so the file might be called www.example.com/somedirectory/somefile.html” etc. Internal links work for collections I've built when I mirrored some of our university pages... I don't know if it will work in your situation though. Let the list know if it does!

Another example of a response would amount to sharing the same experience and how it is solved.

**Example**

Hi I have been using 7.0 with no problems till today. I dl'ed 7.3, followed instructions uninstalled 7.0 from add/delete programs and installed 7.3. Now when i try to start thunderbird it doesn't completely start. In task manager it shows under open processes thunderbird.exe running using 99 % of my cpu resources bogging down everything else but the window with the program never opens and i have to eventually hit close task to get rid of it. I really like tbird and would hate to go back to outlook but i cant seem to get it to work now...ive uninstalled and reinstalled it a couple of times now with no luck

**Response 1**

I've been having a similar problem, I recently re installed thunderbird after having upgraded to SP2 and now the program crashes as soon as I click on the icon, it just brings up the Quality Feedback agent or whatever it's called. I've tried deleting the chrome and extensions folder but it's not helping, any ideas?"

**Response 2**

Having the same problem

A very important trait in FLOSS pertaining to limitations is that these can be overcome as long as the remaining members engage on them. In these environments, the discussion forums prove to make FLOSS communities as communities for sharing and learning [109]. People post problems as well as solutions for the benefit of other members of the community. A solution does not just benefit the help-seeker. The public nature of discussion boards means that they are frequented by people who do not post either questions or answers but benefit from the discussions anyway. This exemplifies the dynamic and volatile nature of FLOSS communities. There were several instances of messages posted a long time after the problem had been solved; where someone else who had the same problem, looked at this discussion and solution and then posted so that other also benefited from it. There were also instances similar to the “I have the same problem” which proved that this process was helpful to not only the people who were asking but to the community at large [109].

In addition to these studies, a different analysis conducted by Steehouder [111] provides some more indications on the type of questions and the contextual needs of interactions in discussion forums. According to this study, the most outstanding result, is the fact that problems are so often described in scenarios. A closer look at these scenarios showed that 25 out of the 31 scenarios use the pronouns I
and/or MY, which can be interpreted as an indication that the problem or question is considered strictly personal. Moreover, 7 out of the 31 scenarios included an adverb of time to intensify the occasional character of the problem [111].

Another perspective into considering questions pertains to their categorization with summarized results in the table given in Figure 48. This information provides more insights on the examples of key phrases used to formulate questions and their categorization as provided below.

<table>
<thead>
<tr>
<th>Question type</th>
<th>Number of opening messages containing this information (N=50)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct questions</td>
<td>15</td>
</tr>
<tr>
<td>Indirect “Is it possible”</td>
<td>13</td>
</tr>
<tr>
<td>questions</td>
<td></td>
</tr>
<tr>
<td>Indirect “Do you know”</td>
<td>7</td>
</tr>
<tr>
<td>questions</td>
<td></td>
</tr>
<tr>
<td>Other indirect questions</td>
<td>15</td>
</tr>
</tbody>
</table>

**Figure 48. Categorization and formulation of questions** [111]

The next study is a culmination of the study started by Singh et al. [109]. This study is very critical as it provides richer insights based on extensive analysis of new data over a longer period of time. Hence, it presents the types of discussions that occur, the types of questions asked and the type of responses that are given.

In collecting data, Singh et al. [112] got a reasonably complete list of the different types of discussions. Consequently they collected twenty threads each from eight different websites making a total of 160 threads. Figure 49 lists the websites for the FLOSS projects they considered in their study.

<table>
<thead>
<tr>
<th>Websites</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

**Figure 49. List of FLOSS projects used for enhanced analysis on messages exchange** [112]

According to Singh et al. [112], these 8 websites were chosen because they represent different types of open source software products - the categorization was adopted from the categorization of open source software products on the Sourceforge.net website. This was done to ensure that different types of products were being used such as browsers, programmers' tools, networking, collaborative software, application software, etc.
Analyzing the data, the authors argue that looking at all 160 threads, discussions in FLOSS can be categorized into five categories, as shown in Figure 50 with their relative frequency. As expected, a large majority (76%) of interactions are problem solving. However, in this data set across eight websites that still leaves almost a quarter of interactions in other categories. Most of the remaining threads are information seeking discussions like “Does this exist?”, “Where can I find ‘x’?”, etc. [112].

<table>
<thead>
<tr>
<th>Types of Discussion</th>
<th>Occurrence (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Solving</td>
<td>76</td>
</tr>
<tr>
<td>Information Seeking</td>
<td>14</td>
</tr>
<tr>
<td>Social Discussion</td>
<td>6</td>
</tr>
<tr>
<td>Feature Request</td>
<td>2</td>
</tr>
<tr>
<td>Information Dissemination</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 50. Types of Discussions in FLOSS [112]

The details provided by Singh et al. [109] as well as the enhanced and more advanced study conducted in Singh et al. [112] suggest that questions posed in FLOSS online discussion forums can be divided into two broad categories: those asked by the help-seekers and those asked by the help-givers [112]. Subsequent analysis gave a number of sub-categories. The help-seekers posted questions about how to get things done with key phrases such as why am I stuck, what is wrong, etc. while the help-givers asked about background information of the help seeker's computer, the task that they were doing, the results of following the steps that they have taken, and so forth [112]. Figures 51 and 52 below give a summary of such categories of questions as evidently found in [112]. As a result of these questions, a number of responses can be provided accordingly. Figure 53 depicts a response typology that is found in these forums categorizing the different types of responses posted on these online discussion forums.
<table>
<thead>
<tr>
<th></th>
<th>Background Information questions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OS?</td>
<td>Browser?</td>
<td></td>
</tr>
</tbody>
</table>

| 2 | Clarification questions | what says phpMyAdmin/libraries/common.lib.php on line 2979 |

| 3 | Give a screenshot | I really can't see and find these icons. Can you screenshot it? |

| 4 | History Details | Any extensions? And have you cleared your downloads history lately? |

| 5 | It works for me, did you try this | Works for me. Did you clear your cache and try it again? If that doesn't work, close Firefox and delete mimeType.rdf from your profile: http://kb.mozillazine.org/Profile_folder# Where_is_my_profile_folder.3 F |

| 6 | Repeat the question | what is your problem then, upload or displaying uploaded table? |

Figure 51. Help-givers questions in FLOSS discussion forums [112]
<table>
<thead>
<tr>
<th>Type of Questions</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Any suggestions / ideas</td>
<td>After the last two updates, I get a regular &quot;Server not found&quot; screen. After I click the &quot;try again&quot; button 4 or 5 times, the page comes up. I have checked everything I can think of and can find no fix. VERY annoying and time consuming. Any suggestions?</td>
</tr>
<tr>
<td>2 Anyone else has this problem</td>
<td>I just updated my firefox and now, every time I open the browser, the toolbar buttons on my Yahoo! toolbar don't load. When I &quot;click here to retry&quot; they come up fine. Does anyone else have this problem?</td>
</tr>
<tr>
<td>3 Can this be done / Is it possible to do this?</td>
<td>&quot;Is it possible force use another charset (Win1251, for example) to make Filezilla Server compatible with older clients (like FAR Manager, for example)?&quot; &quot;I use Moodle as file manager in topic format is it possible to get the topic number larger than 52?&quot;</td>
</tr>
<tr>
<td>4 Comparison</td>
<td>Is it just me or FFx 2.0 RC 3 is a bit faster than Opera?</td>
</tr>
<tr>
<td>5 Does this exist?</td>
<td>Opera seriously needs a better ad-blocker. The ad-blocker that come with Opera 9.02 is horrible IMHO. I have to manually pick what to block! I don't think so; you can't even block some ads.</td>
</tr>
<tr>
<td>6 Feature Request/Exist</td>
<td>Anyone any idea why we are getting a 403 forbidden error code?</td>
</tr>
<tr>
<td>7 Doesn’t work</td>
<td>Media player and Opera not working properly together. When using the next/previous/play etc buttons on my keyboard, they don't work when the Opera window is active</td>
</tr>
<tr>
<td>9 How to</td>
<td>Anyone knows how to bypass the Login screen?</td>
</tr>
<tr>
<td>10 Information needed / Recommendation</td>
<td>Any recommendations for any extension that can support unlimited note pages (preferably in tab style like foxtone) for firefox?</td>
</tr>
<tr>
<td>11 Is it a bug? Is there a workaround?</td>
<td>The paragraph of body text overlaps the heading on my webpage when viewing in Opera. Is this a bug in Opera? Is there a workaround?</td>
</tr>
<tr>
<td>12 What to do?</td>
<td>When I &quot;Print preview&quot; web pages the images dont appear. What setting do I need to Change?</td>
</tr>
<tr>
<td>13 What am I doing wrong?</td>
<td>The page I am trying to edit on my site does not go to the webpage that I can see on the Internet. I can see it under one of the lists in the site manager and when I click on it it appears in the window next to the site manager. Does anyone have any suggestions on what I am doing wrong.</td>
</tr>
<tr>
<td>14 What is happening?</td>
<td>This error should happen once or twice, but it keeps happening, what is going on?</td>
</tr>
<tr>
<td>15 What is wrong?</td>
<td>These are my settings can anyone see what is wrong?</td>
</tr>
<tr>
<td>16 Where</td>
<td>Where to find NVU compatible pre-built template?</td>
</tr>
<tr>
<td>17 Why does this happen? / Why is this happening and how can I fix it?</td>
<td>Opera keeps freezing up and making my computer go slow and i have to push shift alt and delete and kill it. Why does this keep happening? I hyperlinked text in my instruction to the location of the forums. all those links appear to be removed/defaulted to something else/ written in gibberish in the last few days today they are not the same. What can cause this and how can i make sure the links retain their original address?</td>
</tr>
</tbody>
</table>

Figure 52. Help-seekers questions in FLOSS discussion forums [112]
Additional examples for social responses include “thank you”, “you are welcome”, “Search the forum before asking”. To conclude this section, we consider 2 more studies conducted by Lakhani et al. [114] and Barcellini et al. [115].

The main objective of Lakhani et al. [114] was to discuss the idea of user participation and the motivation behind the level of service provided by help-givers in FLOSS considering Apache as a case study. The authors noticed after their experiment that 98% of the effort expended by information providers in fact returns direct learning benefits to those providers. This finding considerably reduces the puzzle of why information providers are willing to perform this task for free. Also, this supports the idea that both information seekers and providers learn from participating and contributing discussions [114].

In Apache, the system takes the form of publicly accessible “newsgroup” discussion forums carried on a segment of the Internet called the Usenet. An Apache user with a question “posts” it on the appropriate Usenet discussion forum. Any interested user can read both the questions and answers that have been posted, and can provide answers or add to the discussion if he or she wishes to do so. Both questions and
answers are typically signed and identified by the e-mail address of the person posting. This is critical in aiding our quest to capturing possible recurring key phrases in FLOSS used for both requesting and providing help. An example of such interaction as taken from [114] is:

Subject: Apache 1.3.1 and FrontPage 98 extensions. A small problem …

Information seeker:
Hi,
I’ve compiled and installed Apache 1.3.1 with mod_frontpage.c. That section seems to be working. I have installed the FrontPage 98 extensions, and that seems to almost be working, but I can’t find any relevant information anywhere about how to solve this problem. I can look at a home page for a user, but I can’t publish to it. Whenever FrontPage tries to connect to the server, this message appears in the error logs:

[Thu Oct 8 10: 13:31 1998] [error] (104) Connection reset by peer: Incorrect permissions on webroot “/usr/local/httpd/htdocs/_vti_pvt” and webroot’s _vti_pvt directory in FrontPageAlias().


I haven’t a clue how to fix it. Any help will be very appreciated, and a reply by e-mail will be noticed more quickly (I’m terrible at remembering to check the newsgroups)

Thanks!

Information Provider 1:
Hi there,
There are two possible causes of your problem:

1. Make sure owner and group are the same and that the directories have the same set of permissions. /home/user/public_html user group/home/user/public_html/_vit_bin www group1 should be: /home/user/public_html user group/public_html/_vit_bin user group

2. Apache-fp utilizes fpexe and mod_frontpage to provide a higher level of security. Part of the mod_frontpage code sets LOWEST_VALID_UID and LOWEST_VALID_GID. Users with UIDs and GIDs less than these values will not be able to run the server extensions. These values are configurable. For more information please check the SERK documentation and the Apache-fp page.

Looking at this scenario, we can identify key phrases used as follows:

<table>
<thead>
<tr>
<th>Information Seeker</th>
<th>Information Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>“but I can’t find”,</td>
<td>“possible causes of your problem”,</td>
</tr>
<tr>
<td>“I haven’t a clue how to fix it”, “Any help will be very appreciated”</td>
<td>“Make sure”,</td>
</tr>
<tr>
<td></td>
<td>“For more information please”</td>
</tr>
</tbody>
</table>

Finally Barcellini et al. [115] provide and describe a number of insights very critical and almost relating to the previous studies. A number of activities are observed and data is analyzed to identify key phrases. An example of such activity is proposal. In this activity, a developer proposes a new design theme or a new design alternative related to an existing theme. To identify this activity Barcellini et al. [115] analyzed expressions like “How about…?”, “Why not…?”, and verbs like “to propose”, and “to add”. Another example of activity is evaluation. For this activity, a developer agrees or disagrees with a proposal or another proposition. This activity is identified by verbs such as “to like”, “to agree”, “to
prefer”; or expressions like “yes”, “no”, “indeed”, “of course”, “great”, “sure”, or “+1”, “−1” (a shorthand for voting for or against a proposal). Sometimes a developer would moderate the discussion or even postpone a task. With key phrases like “I think... probably that is better than xx...”

In light of all these studies, we understand that the information provided can help generate and formulate key phrases. Therefore, our approach in finding these key phrases can be summarized as capable of giving very good results in a way that is robust and makes few assumptions about the content to be analyzed. This is an important consideration given the heterogeneous nature of message contents in FLOSS. In essence, it is based on three key factors:

- Literature on communication in FLOSS communities: As shown above, specifically in [108-115], not much study has been done on FLOSS from a qualitative point of view specifically in terms of communication and message exchanges. While almost the entire literature provides a load of quantitative information on actions, participants, messages sent and received in FLOSS, three studies in particular provide some ground work for our endeavor. We build from the empirical findings in [109,112, 114 and 115] to derive key phrases we need to identify and categorize learning activities in FLOSS.
- Common sense: This refers to the trivial consideration for the meaning of words and activities we have identified.
- Lexical semantics: According to Wikipedia, “Lexical semantics is a subfield of linguistic semantics. It is the study of how and what the words of a language denote. Words may either be taken to denote things in the world or concepts.... One question that lexical semantics explores is whether the meaning of a lexical unit is established by looking at its neighborhood in the semantic net (by looking at the other words it occurs with in natural sentences), or if the meaning is already locally contained in the lexical unit. Another topic that is explored is the mapping of words to concepts.” [116].

Based on these elements, we consider a collection of words in a message either from an email or a message chat to identify a concept (learning activity) that they represent. As lexical semantics also deals with synonyms, and homonyms which are of high relevance to semantic search, making use of Semantic Search is paramount and promises to capture the meaning of message contents as much as possible in identifying activities. Therefore, following this approach we make use of the identified expressions/phrases from all the mentioned sources in order to consolidate and formulate key phrases. For consistency reasons, we table these according to the three stages of learning process in Tables 7-9 that we can refer to as Key Phrase Catalogs for Learning activities. The idea is to map all the learning activities as discussed in the previous chapter with key phrases before we actually process mine our data sets. The state as depicted in the catalogs simply refer to the basic activity as detailed in Chapter 3.
<table>
<thead>
<tr>
<th>STATES</th>
<th>Keywords</th>
<th>PARTICIPANTS</th>
<th>ACTIVITIES</th>
<th>KEYPHRASE/CONDITIONAL ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observation</td>
<td>“problem”, “help”, “error”</td>
<td>NOVICE</td>
<td>FormulateQuestion</td>
<td>If PostQuestion = true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IdentifyExpert</td>
<td>“How did you do this”, “I saw your code”, “I need your help”, “this does not work for me”, “-1”, “is this possible to do this”, “can this be done?”, “very helpful”, “very well”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PostMessage</td>
<td>If IdentifyExpert = true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PostQuestion</td>
<td>“How can I do…?” “How to?”, “don’t understand how”, “could help?”, “what is wrong?”, “my code is not running”, “code not executing”, “question”, “How to”, “what is wrong”, “where can I”, “Any ideas how to solve this problem?”, “I have tried doing”, “search for this”, “but have had little luck”, “any help?”, “any suggestions?”, “everything I could”, “new to the”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CommentPost</td>
<td>“does not work”, “not executing”, “this does not work for me”, “do not know what is wrong with my code”, “here is my code”, “in short my problem”, “step by step”, “details provided”, “works as follows”, “I want it to”, “expect it to”, “my question is like this”, “what I mean”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ContactEstablishement</td>
<td>“can I get your help”, “can you help”, “send question”, “contact details”, “send email”, “send file”, “more details”</td>
</tr>
<tr>
<td>Expert</td>
<td></td>
<td></td>
<td>ReadMessages</td>
<td>If CommentPost = true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReadPost</td>
<td>If CommentPost = true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReadSourceCode</td>
<td>“syntax error”, “maybe you should…”, “it seems to work for me” “do not know what is wrong with the code” or “not sure it can” ) or “running your code”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CommentPost</td>
<td>“system details needed”, “more details needed”, “more problems details needed”, “more details needed of what is on the screen”, “Did it work before?”, “provide exact step by step details”</td>
</tr>
<tr>
<td>ContactEstablishment</td>
<td>“can I get your help”, “can you help”, “send question”, “contact details”, “send email”, “send file”, “more details”</td>
<td>NOVICE</td>
<td>ContactExpert</td>
<td>If SendDetailedRequest = true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SendDetailedRequest</td>
<td>“actually the code is like this…” “I tried this”, “I don’t know how”, “I don’t understand how?” “can you help”, “your help”, “you explain”, “as you asked”, “so my question is”, “I wanted to know”, “what I meant is”, “my screenshot looks”, “I get this error”, “how do I fix this”</td>
</tr>
<tr>
<td>Expert</td>
<td></td>
<td></td>
<td>ContactNovice</td>
<td>If CommentPost/SendFeedback = true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CommentPost/SendFeedback</td>
<td>“does not work”, “not executing”, “maybe you should…”, “it seems to work for me”, “this does not look right” “you need to delete this..”, “the syntax is not correct”, “send me your code”, “what is your problem?”, “this works for me”, “Did it work before”, “I think it should work”</td>
</tr>
</tbody>
</table>
### Table 8. CATALOG OF KEY PHRASES FOR LEARNING ACTIVITIES AT PROGRESSION PHASE

<table>
<thead>
<tr>
<th>STATES</th>
<th>Keywords</th>
<th>PARTICIPANTS</th>
<th>ACTIVITIES</th>
<th>KEYPHRASE/CONDITIONAL ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revert</td>
<td>“on the file”, “your code”, “error”, “same problem”, “help”, “error”</td>
<td>NOVICE</td>
<td>ProvideFeedback</td>
<td>“Still not working”, “but same error”, “It comes up with the following error”, “Many thanks”, “I'm using”, “is it a bug?”, “Is there a workaround?”, “What if”, “is this the same”, “how come”, “what is happening?”, “What am I doing wrong”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PostQuestions</td>
<td>“what if”, “is this the same”, “how come”, “what is happening?”, “What am I doing wrong”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Expert ReviewThreadPosts</td>
<td>I don’t think it will work, working for me, no, -1, don’t think so</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Expert</td>
<td>ReviewThreadCode</td>
<td>problem with your code, error at line, correct it, try this, remove line, rerun it</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SendReply</td>
<td>if still not working, if same error, what happened, As far as I am aware, “send me your code”, “what is your problem?”, “this works for me”, “Did it work before”, “I think it should work”</td>
</tr>
<tr>
<td>Post</td>
<td>“details”, “bug”, “this question”, “file”, “your problem”</td>
<td>NOVICE</td>
<td>PostQuestions</td>
<td>“someone gets the same message”, “why are we getting this error message today”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReplyPostedQuestions</td>
<td>“How about…?, Why not…?”, “actually the code is like this…” “I tried this”, “I don’t know how”, “I don’t understand how?” “can you help, “your help”, “you explain”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SendFeedback</td>
<td>“I don’t think it will work, working for me, no, -1, don’t think so”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Expert</td>
<td>PostQuestions</td>
<td>did you try my solution?, what happened after</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Expert ReviewPostedQuestions</td>
<td>error at line, fix that, try again, troubleshooting, rerun it, try this then tell me, if happened, might happen</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SendFeedback</td>
<td>“maybe try as he said, I think he is, they could, you might consider, can look at, tutorials, read FAQ, read the manual, read wiki, tools, As far as I am aware, I don’t know”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReportBugs</td>
<td>“does not work”, “not executing”, “maybe you should…”, “it seems to work for me”, “this does not look right” “you need to delete this”, “the syntax is not correct”,</td>
</tr>
<tr>
<td>Apply</td>
<td>“tried it”, “working”, “should work”, “error at line”, “created”, “code”, “bug”</td>
<td>NOVICE</td>
<td>ReportBugs</td>
<td>seriously, this needs to be updated, now obsolete, not compatible, “send me your code”, “what is your problem?” “this works for me”, “Did it work before”, “I think it should work”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AnalyseSourceCode</td>
<td>If CommentOnCode = true “As far as I am aware” or “me too” or “experienced the same problem” or “I fixed it by” or “I removed that line in your code” or “then run”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Expert</td>
<td>ReportBugs</td>
<td>If RunSourceCode = true “bug fixed in next version, it is not, it should work, I checked, error at line, fix that, try again, troubleshooting, rerun it</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReplyToPost</td>
<td>If ReplyToPost = true “bug fixed in next version, it is not, it should work, I checked, error at line, fix that, try again, troubleshooting, rerun it</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RunSourceCode</td>
<td>If RunSourceCode = true “bug fixed in next version, it is not, it should work, I checked, error at line, fix that, try again, troubleshooting, rerun it</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CommentOnCode</td>
<td>If RunSourceCode = true “bug fixed in next version, it is not, it should work, I checked, error at line, fix that, try again, troubleshooting, rerun it</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AuthorSourceCode</td>
<td>If RunSourceCode = true “bug fixed in next version, it is not, it should work, I checked, error at line, fix that, try again, troubleshooting, rerun it</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReportBugs</td>
<td>If RunSourceCode = true “bug fixed in next version, it is not, it should work, I checked, error at line, fix that, try again, troubleshooting, rerun it</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AnalyseSourceCode</td>
<td>If RunSourceCode = true “bug fixed in next version, it is not, it should work, I checked, error at line, fix that, try again, troubleshooting, rerun it</td>
</tr>
</tbody>
</table>
### Table 9. CATALOG OF KEY PHRASES FOR LEARNING ACTIVITIES AT MATURATION PHASE

<table>
<thead>
<tr>
<th>STATES</th>
<th>Keywords</th>
<th>PARTICIPANTS</th>
<th>ACTIVITIES</th>
<th>KEYPHRASE/CONDITIONAL ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyze</td>
<td>&quot;on the file&quot;, &quot;your code&quot;, &quot;error&quot;, &quot;same problem&quot;</td>
<td>NOVICE</td>
<td>AnalyzeDiscussions</td>
<td>“not running”, “maybe you could try”, “merge”, “delete”, “merge”, “build failed”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AnalyzeSourceCode</td>
<td>“problem with file”, “error at line”, “correct this”, “try this”, “remove line”, “rerun it”, “build succeeded”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AnalyzeThreadProgression</td>
<td>“working now”, “same error”, “still incorrect code”, “build failed again”, “still a bug”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXPERT</td>
<td>AnalyzeThreadProgression</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AnalyzeSourceCode</td>
<td>error at line, correct it, try this, remove line, rerun it, debug again</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AnalyzeDiscussions</td>
<td>It will not work, “not running”, “maybe you could try”, “merge”, “delete”, “merge”, “add this”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SendFeedback</td>
<td>“this works when”, “I think it should work”, “remove code”, “works for me”, “it should work”, “this is due”, “the cause”, “approved”, “failed”, “succeeded”</td>
</tr>
<tr>
<td>Commit</td>
<td>“file”, “bug”, “patch”, “module”</td>
<td>NOVICE</td>
<td>SubmitDocumentation</td>
<td>If type of commit [file type] = “document”, or “dev doc” [for Source_code repository] and [“Install guide”, “install script”, “certificate”] on issues and [“details at http” on reviews]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SubmitCode</td>
<td>If type of commit [file type] = “code”, or “patch” [for Source_code repository], else key words like “I submitted code”, “my code”, “I wrote these lines”, “my file not working”, “I corrected the new version”, “corrected this module”, “patch working”, [file fixed, updates, imported on issues]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SubmitBugReport</td>
<td>“patch not working”, “missing lines in module”, “missing file”, “module incorrect”, seriously, this needs to be updated, now obsolete, add, remove [in issues], fix, update</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXPERT</td>
<td>ReviewDocumentation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReviewCode</td>
<td>error at line, fix that, try again, troubleshooting, rerun it, try this then tell me, if happened, might happen, this should do the trick, try new version</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReviewReport</td>
<td>“not right document” on [tickets and sourcecode repositories], “looks good to me”, “works for me”, “it should be changed”, “to be fixed”, “to be reviewed” [on reviews]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SendFeedback</td>
<td>“I think it should work”, “remove code”, “works for me”, “it should work”, “this is due”, “the cause”, “it seems to work for me”, “you need to delete this”, “the syntax is not correct”, “simplest way to”, approved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GiveSuggestion</td>
<td>“me too”, “experienced the same problem”, “I fixed it by”, “I removed that line”, “in code”, “then run”, “one way is trying”, “you could try”, “maybe you should”, “can you remove?”</td>
</tr>
<tr>
<td>Activity</td>
<td>Code</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PostCommentOnCode</strong></td>
<td>“code”, “changed”</td>
<td>“error at line”, “fix this line”, “try again”, “troubleshooting”, “rerun it”, “it looks good to me”, “it should work”, “remove lines”, “change data type”, “syntax error”</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ReplyToSuggestion</strong></td>
<td></td>
<td>“I tried that”, “still not working”, “maybe was not clear”, “I meant”, “this code won’t work for me”, “my case is different”, “me too”, “experienced the same problem”, “I fixed it by”, “I removed that line”, “in code”</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WriteSourceCode</strong></td>
<td></td>
<td>If type of commit [file type] = “code”, or “patch” [for Source_code repository], else key words like “I submitted code”, “my code”, “I wrote these lines”, “my file not working”, “I corrected the new version”, “corrected this module”</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ModifySourceCode</strong></td>
<td></td>
<td>If FixBugs = true</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EXPERT</strong></td>
<td><strong>ReportBugs</strong></td>
<td>If RunSourceCode = true</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>AnalyzeSourceCode</strong></td>
<td>If CommentOnCode = true</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>CommentOnCode</strong></td>
<td>problem with your code, error at line, correct it, try this, remove line, fix this line, try again, troubleshooting, rerun it, it looks good to me, it should work, remove lines, change data type, syntax error</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>RunSourceCode</strong></td>
<td>bug fixed in next version, it is not, it should work, I checked, error at line, fix that, try again, after troubleshooting, executed code, build failed</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Revert</strong></td>
<td>“installed”, “added”, “removed”, “run”, “changed”, “code”, “module”, “patch”</td>
<td>“still not working”, “but same error”, “It comes up with the following error”, “Many thanks”, “I’m using”, “it is a bug?”, “Is there a workaround?”, “what if”, “is this the same”, “how come”, “what is happening?”, “What am I doing wrong”</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NOVICE</strong></td>
<td><strong>ProvideFeedback</strong></td>
<td>“Still not working”, “but same error”, “It comes up with the following error”, “Many thanks”, “I’m using”, “it is a bug?”, “Is there a workaround?”, “what if”, “is this the same”, “how come”, “what is happening?”, “What am I doing wrong”</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>PostQuestions</strong></td>
<td>“what if”, “is this the same”, “how come”, “what is happening?”, “What am I doing wrong”, “is it a common bug”, “be removed?”</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EXPERT</strong></td>
<td><strong>ReviewThreadPosts</strong></td>
<td>I don’t think it would work, no, -1, don’t think so</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ReviewThreadCode</strong></td>
<td>problem with your code, error at line, correct it, try this, remove line, rerun it</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>SendReply</strong></td>
<td>“this works for me”, “I think it should work”, “remove code”, “works for me”, “it should work”, “this is due”, “the cause”, “it seems to work for me”, “you need to delete this”, “the syntax is not correct”, “simplest way to”</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Review</strong></td>
<td>“add”, “install”, “add”, “remove”, “run”, “change”, “code”, “module”, “patch”, “troubleshoot”</td>
<td>It is a bug, file not working properly, bug fixed in next version, it is not, it should work, I checked, error at line, fix that, try again, troubleshooting, rerun it, build failed</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NOVICE</strong></td>
<td><strong>ReportBugs</strong></td>
<td>“Can you describe the bug?”, I don’t think it would work, no, -1, don’t think so, 1, agreed, also another suggestion</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ReviewPosts</strong></td>
<td>If ReportBugs = true</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ReviewSourceCode</strong></td>
<td>If ReviewPosts = true</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>EXPERT</strong></td>
<td><strong>ProvideFeedback</strong></td>
<td>“this works for me”, “I think it should work”, “remove code”, “works for me”, “it should work”, “this is due”, “the cause”, “it seems to work for me”, “you need to delete this”, “the syntax is not correct”, “simplest way to”, build failed</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ReviewSourceCode</strong></td>
<td>If ReportBugs = true</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>ReportBugs</strong></td>
<td>bug fixed in next version, it is not, it should work, I checked,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Message</td>
<td>Action</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>error at line, fix that, try again, troubleshooting, rerun it, build failed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Review Posts</th>
</tr>
</thead>
<tbody>
<tr>
<td>If ProvideFeedback = true</td>
</tr>
</tbody>
</table>
4.4 Conclusion

We presented the context in which Process Mining is applicable to our analysis in this chapter. We started by describing the general landscape in FLOSS and the basic components or elements that we called preliminaries that have to be satisfied in order to make FLOSS data ready for Process Mining. We pointed out that this readiness consists mainly in organizing these FLOSS data and constructing Event Logs. Furthermore, in a step-by-step fashion, we described our approach on how to produce these Event Logs through generating key phrases and making use of Semantic Search in SQL.

In the next chapter, the emphasis is on the application of this proposed approach. Specifically, we discuss the attainment of technical requirements. We introduce and describe the data sets, we configure them for Semantic Search and introduce the algorithms that construct and build the Event Logs before presenting the results of the experiments in subsequent chapters.
CHAPTER 5:
PROCESS MODELS FOR LEARNING PROCESS: TECHNICAL SPECIFICATIONS

5.1 Introduction
In the previous chapter, we started a concrete discussion regarding the application and implementation of Process Mining as part of our analysis. Having set the tone for the construction of Event Logs, the next step is to apply our approach, produce the logs and analyze them as needed. The focus in this chapter shifts to this objective. We complete this discussion by looking at the technical specifications.

Specifically, we discuss the last layer of prerequisites leading to the generation of Event Logs and application of Process Mining techniques to these logs. The key to this empirical analysis is to make sure that we present a systematic way of mining the FLOSS repositories we have identified and producing the evidence of learning processes in these logs. This evidence, in the form of traces primarily accounting for users’ activities while performing their tasks in FLOSS communities is represented through process models. Therefore, it is critical that we highlight that throughout this thesis, any reference to process maps entails process models. These two synonymous terms can be used interchangeably in most cases with a small level of demarcation. On the one hand, a process map is a pictorial representation of the sequence of actions that comprise a process. It provides an opportunity to learn about work that is being performed and provides a reference to discuss how things get done [166-170]. As evidence of learning activities, the graphical representation we get directly from Event Logs is described in these chapters through process maps. On the other hand, the term “process models” is used to describe a more generalized conceptual depiction of the learning behavior in FLOSS. In the final chapters, when talking about conformance analysis and final models, we refer more specifically to Petri net models. Nevertheless, in this chapter and the next three chapters, the two terms process models and process maps are used interchangeably.

This chapter describes the data source from which we considered our FLOSS repositories, the configuration of these repositories to enable Semantic Search, the identification of learning activities based on key phrases catalogs and classification rules expressed through pseudocode as well as the Process Mining tool used in this regard.

5.2 Data Source
A myriad of FLOSS platforms can be found including Sourceforge [171], Freshmeat [172], Savannah.gnu [172], PRIDE [175], GitHub [174] and FLOSSmole [46]. A review of these platforms yields a few shortcomings with regards to the availability of data we require for our analysis. The most notable was incompleteness and insufficiency of available data. Most projects found in these repositories store rather different types of metadata regarding the actual FLOSS records such as overall statistical information about users, their activities and the total number of codes aggregated according to the corresponding projects, without providing adequate detailed data about these projects themselves. Furthermore, some of these platforms only exist historically as many projects are now inactive and discontinued.

Therefore, we considered a more suitable FLOSS platform called OpenStack [163]. According to Wikipedia, “OpenStack is a free and open-source software cloud computing software platform. Users primarily deploy it as an infrastructure as a service (IaaS) solution. The technology consists of a series of interrelated projects that control pools of processing, storage, and networking resources throughout a
**data center**—which users manage through a web-based dashboard, command-line tools, or a RESTful API. OpenStack.org releases it under the terms of the Apache License” [163].

The main reasons behind the choice to use this platform are twofold. Primarily, the availability of relevant data needed to carry out our experiments across the three learning process phases. Lastly, we considered this platform because it is more recent and still active and therefore offers a more appealing option. Data is available and downloadable from http://activity.openstack.org/dash/browser/data/db/. We downloaded the version of data available until May/June 2014.

OpenStack contains five main repositories that we have retrieved in order to conduct our analysis. These repositories that we sometimes refer to as data sets include Mailing archives, Internet relay Chat (IRC) messages, Source Code, reviews and Bug reports. Each of these repositories is described in details in the subsequent chapters as we produce and explain the process maps. Nevertheless, in the next section we demonstrate Semantic Search was enabled on these repositories.

### 5.3 Data Analysis

There are two main steps involved in terms of data analysis. The first step is the enabling of the Semantic Search capabilities on our data sets and the last step is about identifying learning activities through a set of rules expressed as Pseudocode.

#### 5.3.1 Enabling Semantic Search

As we described in Chapter 4, the configurations for Semantic Search on these repositories follows a three step process: first creating a catalog, then a unique index on the concerned table and lastly enabling full-text and semantic indexing on the identified attributes.

1. **Mailing archives**

   USE openstack_mailing_lists;
   GO
   CREATE FULLTEXT CATALOG MailingFTCat;

   Before we create full-text indexes on identified tables in the repository, we need to make sure they have a unique single-column, non-nullable index. We achieve that through a `CREATE INDEX` statement that creates a unique index, `ui_mail`, on the message_ID column of the `messages` table as follows:

   CREATE UNIQUE INDEX ui_mail
   ON messages(message_ID);

   Having created a unique key, we can now create a full-text index and enable semantic indexing on the `messages` table specifying the concerned attributes as follows:

   CREATE FULLTEXT INDEX ON messages
   (subject
    Language 2057
    Statistical_Semantics,
    message_body
    Language 2057
    Statistical_Semantics)

   KEY INDEX ui_mail ON MailingFTCat
   WITH CHANGE_TRACKING AUTO;

2. Internet Relay Chat (IRC) messages

USE openstack_internet_relais;
GO
CREATE FULLTEXT CATALOG InternetFTCat;

Before we create full-text indexes on identified tables in the repository, we need to make sure they have a unique single-column, non-nullable index. We achieve that through a CREATE INDEX statement that creates a unique index, `ui_internet` as follows:

CREATE UNIQUE INDEX ui_internet
ON irclog(id);

Having created a unique key, we can now create a full-text index and enable semantic indexing on the `irclog` table specifying the concerned attributes as follows:

CREATE FULLTEXT INDEX ON irclog
  (message
   Language 2057
   Statistical_Semantics
  )
  KEY INDEX ui_internet ON InternetFTCat
  WITH CHANGE_TRACKING AUTO;
GO

3. Reviews

USE openstack_reviews;
GO
CREATE FULLTEXT CATALOG ReviewFTCat;

We creates a unique index, `ui_review` as follows:

CREATE UNIQUE INDEX ui_review
ON comments (id);

Having created a unique key, we can now create a full-text index and enable semantic indexing on the `comments` table specifying the concerned attributes as follows:

CREATE FULLTEXT INDEX ON comments
  (text
   Language 2057
   Statistical_Semantics
  )
  KEY INDEX ui_review ON ReviewFTCat
  WITH CHANGE_TRACKING AUTO;
GO

Since we need to use a second source here, we specify the same parameters on its attributes:
CREATE UNIQUE INDEX  ui_issue
ON  issues (id);

Having created a unique key, we can now create a full-text index and enable semantic indexing on the comments table specifying the concerned attributes as follows:

CREATE FULLTEXT INDEX ON issues

(  
  summary  
  Language 2057  
  Statistical_Semantics  
  )  

KEY INDEX  ui_issue ON ReviewFTCat
  WITH CHANGE_TRACKING AUTO;

GO

4.  Source code

USE openstack_source_code;  
GO
CREATE FULLTEXT CATALOG SourceFTCat;

We create a unique index,  ui_source  as follows:

CREATE UNIQUE INDEX  ui_source
ON  scmlog (id);

Having created a unique key, we can now create a full-text index and enable semantic indexing on the comments table specifying the concerned attributes as follows:

CREATE FULLTEXT INDEX ON scmlog

(  
  message  
  Language 2057  
  Statistical_Semantics  
  )  

KEY INDEX  ui_source ON SourceFTCat
  WITH CHANGE_TRACKING AUTO;

GO

5.  Bug Reports

USE openstack_tickets;  
GO
CREATE FULLTEXT CATALOG TicketFTCat;

We create a unique index,  ui_ticket  as follows:

CREATE UNIQUE INDEX  ui_ticket
ON  comments (id);
We now create a full-text index and enable semantic indexing on the comments table specifying the concerned attributes as follows:

```sql
CREATE FULLTEXT INDEX ON comments
  (text
   Language 2057
   Statistical_Semantics
  )
  KEY INDEX ui_ticket ON TicketFTCat
  WITH CHANGE_TRACKING AUTO;
GO
```

Since we need to use a second source here, we specify the same parameters on its attributes:

```sql
CREATE UNIQUE INDEX ui_issue_ticket
ON issues (id);
```

Having created a unique key, we can now create a full-text index and enable semantic indexing on the comments table specifying the concerned attributes as follows:

```sql
CREATE FULLTEXT INDEX ON issues
  (summary
   Language 2057
   Statistical_Semantics,
   description
   Language 2057
   Statistical_Semantics
  )
  KEY INDEX ui_issue_ticket ON TicketFTCat
  WITH CHANGE_TRACKING AUTO;
GO
```

### 5.3.2 Rules for identification of learning Activities

The key in Process Mining is to identify events. An event, as already defined in this thesis, is a tuple made up essentially of case ID, performer, activity and any other relevant attributes. In our case, we include the state, date as well as the role (Novice, Expert). Other ingredients include the catalogs described in the previous chapter as well as our data sets.

The rules expressed in pseudo code below are critical at this step as they provide a complete description of how we retrieved our Event Logs, constituted of sets of all identified events.

An event $E$ is a sextuple $(t, a, p, d, s, r)$ such that:
- $t$ is the case in the event and can be either a topic on emails or an issue number on code and bug reports;
- $a$ is the activity;
- $p$ is the participant;
- $d$ is the relevant date of occurrence;
- $s$ is the state of in the learning process phase and
- $r$ is the participant’s role in the process.
We refer to the catalogs introduced earlier to retrieve the mappings between key phrases, activities, states and participants. Let $c_1$, $c_2$ and $c_3$ denote catalogs respectively for Initiation, Progression and Maturation phases. We distinguish between key phrases for activities and states. We therefore shall refer to key phrases for states as $gl\_key$ (global keys) while the key phrases that help distinguish activities will be referred to as $lc\_key$ (local keys).

We can also set catalogs as sextuples $(C,c_i, gl\_key, state, lc\_key, activity, role)$ such that:
- $C$ is the set of all our catalogs;
- $c_i \in C$ is a single catalog;
- $gl\_key$ is the key phrase for the identification of a state;
- $state$ is the state as it appears in the catalog;
- $lc\_key$ is the key phrase used to identify an activity;
- $activity$ is the corresponding activity in the catalog and
- $role$ is the role as it appears in the catalog.

We construct our Event Logs based on some rules and steps as follows.
— Pseudo Code Rules for Initiation phase

**INPUT:** Considering \( c_i \) as input

**PROCEDURE:** ConstructLog \((c)\)

- Set \( g_{\text{total}} \leftarrow \text{COUNT}(gl\_key) \)
- Set \( l_{\text{total}} \leftarrow \text{COUNT}(lc\_key) \)
- Set \( a' \) a derived activity
- Set \( E' \) a derived event, \( s' \) derived state, \( r' \) a derived role

Do While emails still in data set

For \( i \) in 1 to \( g_{\text{total}} \)

If message\_body matches \( gl\_key \) then

For \( j \) in 1 to \( l_{\text{total}} \)

If message\_body matches \( lc\_key \) then

\[ t \leftarrow \text{topic} \]
\[ a \leftarrow \text{activity} \]
\[ p \leftarrow \text{sender} \]
\[ d \leftarrow \text{date} \]
\[ s \leftarrow \text{state} \]
\[ r \leftarrow \text{role} \]

Add \( E(\ t, \ a, \ p, \ d, \ s, \ r) \) to \( \text{Initiation\_log} \)

---Get the topic
---Get the activity
---Get the sender
---Get the date
---Get the state
---Get the role

Endif

Endfor

Else

\[ a' = \text{‘Participate in Discussions’} \]

---A non-identifiable activity

\[ s' = \text{‘Participation’} \]

\[ r' = \text{‘Inactive’} \]

\[ E' = (t, \ a', \ p, \ d, \ s', \ r') \]

Add \( E(\ t, \ a, \ p, \ d, \ s, \ r) \) to \( \text{Initiation\_log} \)

EndIf

Endfor

Enddo

Open \( \text{Initiation\_log} \)

Do While (not end-of-file)

If \( a = \text{‘PostQuestion’} \) then

\[ a' = \text{‘FormulateQuestion’} \]

Elseif \( a = \text{‘IdentifyExpert’} \) then

\[ a' = \text{‘PostMessage’} \]

Elseif \( a = \text{‘CommentPost’} \) then

\[ a' = \text{‘ReadPost’} \]

Elseif \( a = \text{‘SendDetailedRequest’} \) then

\[ a' = \text{‘ContactExpert’} \]

Elseif \( a = \text{‘CommentPost/SendFeedback’} \) then

\[ a' = \text{‘ContactNovice’} \]

Else

do nothing

EndIf

\[ E' = (t, \ a, \ p, \ d, \ s, \ r) \]

Add \( E(\ t, \ a, \ p, \ d, \ s, \ r) \) to \( \text{Initiation\_log} \)

Enddo

**Return** \((\text{Initiation\_log})\)

**ENDPROCEDURE**
— Pseudo Code Rules for Progression phase

**INPUT:** Considering $c_2$ as input

**PROCEDURE:** ConstructLog ($c_i$)

Set $g_{total} \leftarrow \text{COUNT}(gl\_key)$

Set $l_{total} \leftarrow \text{COUNT}(lc\_key)$

Set $a'$ a derived activity

Set $E'$ a derived event, $s'$ derived state, $r'$ a derived role

Do While emails still in data set

For $i$ in 1 to $g_{total}$

If message_body matches $gl\_key$ then

For $j$ in 1 to $l_{total}$

If message_body matches $lc\_key$ then

---Get the topic

---Get the activity

---Get the sender

---Get the date

Add $E (t, a, p, d, s, r)$ to $Progression\_log$

---add this event to the log

EndIf

Endfor

Else

$a' = \text{\textquoteleft}Participate in Discussions\textquoteright$ ---A non-identifiable activity

$s' = \text{\textquoteleft}Participation\textquoteright$

$r' = \text{\textquoteleft}Inactive\textquoteright$

$E' = (t, a', p, d, s', r')$

Add $E (t, a, p, d, s, r)$ to $Progression\_log$

EndIf

Endfor

Enddo

Open $Progression\_log$

Do While (not end-of-file)

If $a = \text{\textquoteleft}CommentOnCode\textquoteright$ then

$a' = \text{\textquoteleft}AnalyseSourceCode\textquoteright$

Elseif $a = \text{\textquoteleft}ReplyToPost\textquoteright$ then

$a' = \text{\textquoteleft}ReportBugs\textquoteright$

Elseif $a = \text{\textquoteleft}RunSourceCode\textquoteright$ then

$a' = \text{\textquoteleft}CommentOnCode\textquoteright$

Elseif $a = \text{\textquoteleft}RunSourceCode\textquoteright$ then

$a' = \text{\textquoteleft}AnalyseSourceCode\textquoteright$

Else

do nothing

EndIf

$E' = (t, a, p, d, s, r)$

Add $E (t, a, p, d, s, r)$ to $Progression\_log$

Enddo

**Return** ($Progression\_log$)

**ENDPROCEDURE**
## Pseudo Code Rules for Maturation phase

**INPUT:** Considering $c_i$ as input

**PROCEDURE:** ConstructLog ($c_i$)

Set $g_{total} \leftarrow \text{COUNT}(gl_key)$

Set $l_{total} \leftarrow \text{COUNT}(lc_key)$

Set $a'$ a derived activity

Set $E'$ a derived event, $s'$ derived state, $r'$ a derived role

Do While issues still in data set

For $i$ in 1 to $g_{total}$

If issue_description matches $gl_key$ then

For $j$ in 1 to $l_{total}$

If message_body matches $lc_key$ then

$t \leftarrow \text{issue}$

$a \leftarrow \text{activity}$

$p \leftarrow \text{sender}$

$d \leftarrow \text{date}$

$s \leftarrow \text{state}$

$r \leftarrow \text{role}$

Endfor

elsif type of commit [file type] = “document” or “dev doc” then

$a \leftarrow \text{SubmitDocumentation}$

elsif type of commit [file type] = “code”, or “patch” then

$a \leftarrow \text{SubmitCode}$

Add $E(t, a, p, d, s, r)$ to Maturation_log

EndIf

Endfor

Else

$a' = \text{‘Participate in Discussions’}$

$s' = \text{‘Participation’}$

$r' = \text{‘Inactive’}$

$E' = (t, a', p, d, s', r')$

Add $E(t, a, p, d, s, r)$ to Maturation_log

EndIf

Enddo

Open Maturation_log

Do While (not end-of-file)

If $a = \text{‘FixBugs’}$ then

$a' = \text{‘ModifySourceCode’}$

Elseif $a = \text{‘RunSourceCode’}$ then

$a' = \text{‘ReportBugs’}$

Elseif $a = \text{‘CommentOnCode’}$ then

$a' = \text{‘AnalyzeSourceCode’}$

Elseif $a = \text{‘ReviewPosts’}$ then

$a' = \text{‘ReviewCommentContents’}$

Elseif $a = \text{‘ReportBugs [E-Rv]’}$ then

$a' = \text{‘ReviewSourceCode’}$

Else

do nothing

EndIf

$E' = (t, a, p, d, s, r)$

Add $E(t, a, p, d, s, r)$ to Maturation_log

Enddo

Return (Maturation_log)

ENDPROCEDURE

ENDPROCEDURE
5.4 DISCO as a tool for Process Mining

In order to process mine these records, we choose an appropriate tool for analyzing the identified events and that can provide efficient visualizations to demonstrate the workflow of occurrence of activities in these processes. We consider Disco [164]. Disco stands for Discover Your Processes. It is a toolkit for Process Mining that enables the user to provide a preprocessed log specifying case, activities, originator and any other attributes. The tool performs automatic process discovery from the log and outputs process maps as well as relevant statistical data.

In essence, Disco applies Process Mining techniques in order to construct process models based on available logging data that is turned into an Event Log. This logging data is all the details about transactions that can be found in log file or transaction databases. Therefore, an Event Log can take a tabular structure containing all recorded events that relate to executed business activities.

For illustrative purposes, we consider a simple purchasing process [165]. The idea is that the process represents a series of tasks (activities) starting from ordering of goods. Then, these goods get delivered and after their reception, the supplier issues an invoice which is finally paid by the company that ordered the goods [165]. The corresponding Event Log in this case excluding participants can be seen in Figure 54. The original Event Log contains data about 5 cases. However, due to space constraints we show a snippet of the Event Log in Figure 54 with the first 2 cases only. Using Process Mining techniques, Disco can turn such an Event Log into a process model as seen in Figure 55.

<table>
<thead>
<tr>
<th>Case ID</th>
<th>Event ID</th>
<th>Timestamp</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
<td>01.01.2013</td>
<td>Order Goods</td>
</tr>
<tr>
<td>1</td>
<td>1001</td>
<td>10.01.2013</td>
<td>Receive Goods</td>
</tr>
<tr>
<td>1</td>
<td>1002</td>
<td>13.01.2013</td>
<td>Receive Invoice</td>
</tr>
<tr>
<td>1</td>
<td>1003</td>
<td>20.01.2013</td>
<td>Pay Invoice</td>
</tr>
<tr>
<td>2</td>
<td>1004</td>
<td>02.01.2013</td>
<td>Order Goods</td>
</tr>
<tr>
<td>2</td>
<td>1005</td>
<td>01.01.2013</td>
<td>Receive Goods</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 54. An example of Event Log Structure**

![Diagram of the purchasing process](image-url)

**Figure 55. Model for a purchasing process**

The model in Figure 55 provides a representation of the process corresponding to the Event Log in Figure 54 for all 5 cases. The process model in Figure 55 can be interpreted as a dependency graph depicting the
dependencies between different activities. For example one can note that A was followed three times by B and two times by C [165].

In Chapters 6-8, we constructed the Event Logs using the algorithms expressed through the pseudocode described in Section 5.3. Making use of Semantic Search, we identified the corresponding learning activity every time a user performs an action as recorded in our repositories. As part of the process discovery task, we analyze these Event Logs using Disco in order to trace learning processes. The resulting process maps are documented and described through statistical information as provided by Disco. In Disco, each process model can be represented with two different set of metrics on the basis. These metrics include frequency and performance.

The main objective of the frequency metrics is the depiction of how often certain parts of the processes have been executed. We can distinguish three levels of frequency: absolute frequency, case frequency and maximum repetitions. The roles of such levels of frequency can be summarized as follows:

- **Absolute frequency**: When this value is associated with an activity, it indicates how many times in total an activity has been performed, while it also gives an indication of how often activities moved from one point to another on the edge (path), or how often that particular path has been “travelled” on throughout the whole process;
- **Case frequency**: This allows to ignore repetitions that might have occurred with activities, and only shows the relative number of cases that have passed through a specific path;
- **Max. Repetitions**: This provides the maximum number of repetitions within a case.

Unlike frequency, the performance metrics provide details about the time at different levels and paths during the execution process. This can also be aggregated to three different levels:

- **Total duration**: This is metric represents the accumulated duration (summed up over all cases) for the execution of each activity including the delays on each path;
- **Mean duration**: This is the average time spent within and between activities;
- **Max. duration**: This is an indication of the largest execution times and delays that were measured during the process execution.

These metrics can also be provided at once for a path of an activity if needed. We give more details in this regard in the next chapter.

### 5.5 Conclusion

The undertaking to mine FLOSS repositories with the objective to identify learning processes yields process models that we sometimes refer to as process maps. In this chapter, we discussed the foundational technical imperatives required to produce empirical evidence of the traces of learning activities in FLOSS as well as the flow according to which these activities are executed. We described as part of these requirements, the identification rules that guide the construction of Event Logs and briefly introduced the tool we make use of to conduct our experiments. As we produce process maps with Disco, we break down and represent these process maps according to the three levels of learning phases.

We devote the next three chapters to the mining of Openstack repositories for the Initiation, Progression and Maturation phases respectively. In each of these, we detail and describe the data pertaining to the analysis for all participants involved in the learning processes including the Expert and Novice as well as the results of the analysis.
CHAPTER 6:
PROCESS MAPS FOR INITIATION PHASE:
EMPIRICAL RESULTS

6.1 Introduction
In the previous two chapters we laid the foundation needed to conduct our experiments in order to provide empirical results. These results are paramount as they speak to the core of this research by giving evidentiary indications of the existence of learning processes in FLOSS (process discovery) and the extent to which they are different to the current representation (conformance analysis). Therefore, we describe the results of the experiments first as process maps from this chapter to Chapter 8 and then perform conformance analysis in Chapter 9. The description of process maps, as discovered, is given according to Initiation, Progression and Maturation phases in Chapter 6, Chapter 7 and Chapter 8 respectively.

In this chapter, we look at the processes discovered as they relate to the Initiation phase. Principal activities in this first phase of the learning process are generally about observing and making contacts. Ideally, this step constitutes an opportunity for the Novice to ask questions and get some help depending on the requests while the Expert intervenes to respond to such requests. In order to conduct our analysis, we set to identify the most appropriate repository in this regard. The main criteria in making such a decision lies on the existence of some form of communication between FLOSS members on any candidate repository. Therefore, Mailing Archives and Internet Relay Chat messages are the two datasets we have considered to track these activities and explain their flow of occurrence. This is represented through process maps that we call also refer to as process models. It is important, as a reminder, to note that these two terms (process map and process model) are largely synonymous and are used interchangeably throughout this thesis. However, in this chapter and the subsequent two, we have adopted the term process map as it provides a pictorial representation of processes as they occur. Although these process maps are models, we reserve the connotation “process model” to a more formal specification of the learning behavior through either a-priori models (Workflow nets) or final models (Petri nets).

Before we describe the repositories and present the experimental results, it is important to note that in most cases, the way we describe the results about both participants of the learning process, Novice and Expert, is the same. Therefore, it is possible that the explanations in both cases would differ solely on statistical figures on the discovered process maps but keep the same wording. This is simply because in such cases, it is suitable to use a single way of describing the same properties of the process maps for both participants.

6.2 Mailing Archives’ Messages
The first repository where we potentially can find learning activities is the Mailing archives dataset. This database is made up of 7 tables that store data pertaining to compressed files (Source code file, Bug reports etc.), the mailing lists as per group discussions and topic of interests, and the number of messages exchanged as well as details of the individuals involved in these exchanges as shown in the table below.
This repository contains exactly 54762 emails exchanged between 3117 people who are registered on 15 different mailing lists. These emails were sent during a period of time spanning from 2010 to 2014. The first message recorded or the very first email sent was at 10:34:23 on the 11th of November 2010 while the last email considered was sent at 12:16:22 on the 6th of May 2014. The length of the messages considered is on average of 3261 characters, the longest email was of 65535 characters while the shortest message yields a single character length.

We analyzed this dataset and produced the process maps representing the occurrence of learning activities in email messages and this output consists of three process maps. The first process map represents the full learning process while the last two are representations of activities undertaken by the Novice and Expert respectively. In order to fully grasp the processes as well as the underlying paths, two main views are available to present the process maps. We consider the metrics introduced in Chapter 6, namely frequency and performance. In addition to these views, we provide relevant statistics in order to explain the extent of learning activities in FLOSS communities.

In order to provide an overall overview of how FLOSS members interact and exchange knowledge, we specifically produce global process maps with absolute frequency and total duration as main metrics. However, as we break down the process maps according to individual Novice and Expert roles, we consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.

<table>
<thead>
<tr>
<th>Tables</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo.compressed_files</td>
<td>401</td>
</tr>
<tr>
<td>dbo.mailing_lists</td>
<td>15</td>
</tr>
<tr>
<td>dbo.mailing_lists_people</td>
<td>4434</td>
</tr>
<tr>
<td>dbo.messages</td>
<td>54762</td>
</tr>
<tr>
<td>dbo.messages_people</td>
<td>54762</td>
</tr>
<tr>
<td>dbo.people</td>
<td>3117</td>
</tr>
<tr>
<td>dbo.people_uppeople</td>
<td>3117</td>
</tr>
</tbody>
</table>

Table 10. Mailing Archives Description
Figure 56. Process Map for Initiation Phase – Per frequency of occurrence in Mailing Lists
Figure 57. Process Map for Initiation Phase – Per Duration of Activities in Mailing Lists
Making use of Disco, we analyzed our Event Log and the resulting model depicted in Figures 56 and 57, represents a map of activities as they occur during the first phase of the learning processes. In a nutshell, it pens out a workflow representation of how learning activities are executed. Some important components can be noticed in the model. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed in Figure 56, while they represent the total duration of occurrence in Figure 57.

This process map or any for that matter can be represented with various degrees of annotations as defined by the needs of the investigation. But for our purpose as defined by the objectives of this study, we retained the topic of emails (or thread), the message itself, the people involved in exchanging these emails, the resulting activities and classification of where they fall in our defined learning curve.

Additional details regarding the numerical measures and occurrence flow of activities such as events over time, active cases during this given period of time, case variants, the number of events per case as well as case duration could be plotted as needed. However, for simplicity and effectiveness, we represent only major statistical details that are most representative of the presence, impact and occurrence of learning activities in FLOSS over the chosen period of time. We explain this information in the following successive figures produced using Disco.

Figure 58. Events over time

Figure 58 depicts a representation of the time frame for the occurrence of our process events is given. The right hand side of the figure gives some details with regards to input data used in the plot. The log timeline on the horizontal axis represents the total timeframe covered by our dataset (from earliest to latest timestamp observed). The figure shows the level of activity in the process by plotting the number of performed activities in the process on the vertical axis.

The process map representing the Initiation Phase of the learning process started on the 11th of November 2010 and ended on the 6th of May 2014. During this time, we note that a total of 123401 events were generated. An event represents a tuple made up of the case (in this context, the discussion topic), the email senders as well as the relevant learning activities. With about 565 cases, a total of 14 activities are executed with an average time per case of 69.9 days while the median duration is of 57.8 days.

In Figure 59, we give a representation of the case duration metric that shows the throughput time of the process from the very beginning (start of first activity) to the very end (completion of last activity). For example, in Figure 59 one can see that there are a number of short-running cases (345 cases last up to 12 days and 17 hours) and then, towards the right, a smaller number of very long-running cases (up to 3 years or more) which can be explained by the volatile nature of discussions in FLOSS. This happens sometimes when a question is posed or a discussion started but the reactions from people are delayed.
Additionally, the fact that the majority of cases (about 61%) take up to 12 days and 17 hours is indicative of how much interactions people create and engage in and eventually culminate into learning processes. For a period of over 3 years, in the majority of cases, people participated in discussions and reacted to information exchange relatively fast.

The next critical element describing the process maps, is activity. Learning activities can help determine the flow and occurrence of the learning process as well as the participants. In Figure 60, details about activities can be observed.

The statistics about activities depict performance metrics with regard to the role and impact factor for each single activity in the process. Five specific indicators shown at the right hand side of the chart include the number of activities (14), the minimal frequency (58) which speaks of how often the least frequent activity has occurred, the mean frequency which represents the average occurrence of an activity (8,814.36), the maximal frequency (39,829) which depicts how often the most frequent activity has occurred and finally the frequency std. deviation (16639.96) which is simply the standard deviation for the frequency of activities.

Looking at the figure above, we can observe that the activity “Comment Post” by the Novice is executed 32, 28% of the process occurrence. This activity is triggered by a set of key phrases including “does not work”, “not executing”, “this does not work for me”, “do not know what is wrong with my code”, “here is...
my code”, “in short my problem”, “step by step”, “details provided”, “works as follows”, “I want it to”, “expect it to”, “my question is like this”, “what I mean”. These key phrases identify messages (emails) sent by the Novice expressing a number of feelings, opinions or needs. These can range from expressing the fact that whatever is being discussed did not work for him, or give clarification concerning his initial request by rephrasing the question (concern) or even showing the error to indicate the kind of problem he/she is facing. The following activity, “Post Message” is closely related to the third one (Identify Expert) because it is its direct resultant.

The activity “Identify Expert” occurs 31.87 % of the process occurrence. This activity is triggered by a set of key phrases, including “How did you do this” or “I saw your code” or “I need your help” or “this does not work for me” or “-1” or “is this possible to do this” or “can this be done?” or “very helpful” or “very well” or “can you explain”. These key phrases defined messages that assume that the person whose message is being responded to has a certain level knowledge of the question or a particular subject. It would appear that if they posted a response, or some step by step indications on how to solve a particular problem, a lot of people would comment on it, some with the need to ask for clarification, others with sense of acquiescence, some with the need to make contact for further exchange. We indicate that “Post Message” is a resultant of this activity based on the assumption that when people have posted messages expressing the need to get help or having identified someone more knowledgeable about their problems, there is a need to distinctively document that through “Post Message” activity.

The rest of activities are proportionally and relatively less because, we can assume at this point, that less interaction has occurred. After a Novice has posted a message, there has been a reaction and hence the rest of the process unfolds. Also to note here is the presence of the activity “Participate in Discussions” which is an activity that the participants engage into beside the ones we have predefined and looked for in the dataset. As we semantically searched the data, the analysis is that when the emails meet our predefined criteria, they can be classified according to identified activities, otherwise they should be labeled as a generic activity called “Participate in Discussions”. This label represents all other activities that possibly take place in FLOSS during this first phase of learning process other than the one identified in our specification. We shall discuss more about this during conformance analysis in Chapter 9.

In the next two subsections we focus on each learning participant’s role, Novice and Expert. But before then, it is imperative that for consistency we provide global indicators of our discovered process map with regards to three other attributes namely learning participants, states as well as roles in Figures 61-63.

In Figure 61, just like with activities, overall performance facts about participants in the process are provided. It should be noted that during this 3 and half year time frame, a total of 2454 FLOSS members took part in this process (in its first phase) where they sent at least 1 message, a maximum of 4010 activities per participant at an average of 50.29 activities for a 183.05 standard deviation.
While we cannot list all the participants, the Pareto graph in the figure exemplifies how some details about a single participant can be obtained. The displayed yellow line shows the cumulative, relative sum (how much out of 100%) for the values below. In this case, Sascha Peilicke has executed about 82 activities and when added to the previous sum, it helps reach 78.9% of all executed activities. This in essence shows the progression in terms of each participant contributions for the complete process map.

In Figures 62 and 63, we show how the model has been built in the context of learning states as well as roles (Novice, Expert, and Inactive). In Figure 62 specifically, it can be noted that three main states were registered in the model namely the Observation, ContactEstablishment and Participation. While the first two are recorded in our specification, the last state is the classification for all activities and participants that do not fall in our initial categorization. As with the activities, when it is found that a message's content specifies a different activity other than the one indicated in the specification, we conclude that the resulting activity is part of the “Inactive” state as the performer in this context is excluded from the learning process.

![Figure 62. Details about States](image)

In Figure 62, the majority of activities goes to the observation state (122524 in total) with an average frequency of 41,133.67 while the standard deviation is 70,486.8. The figure depicts that the second state has about 754 activities getting the cumulative sum to just 99.9%.

![Figure 63. Details about Roles](image)
Finally, we can note how participants in quest for knowledge justifiably claim the majority of activities with a total of 122,838 amounting to 99.54% of all executed activities at this point in contrast with Experts who intervene at a lower rate of 0.36% with 440 activities, ahead of people doing something other than exchanging knowledge with 123 activities.

We provide some more insights about what can justify this discrepancy in the next two subsections as we unpack activities flow for Novices and Experts respectively. As indicated earlier, we shall consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.
Figure 64. Process Map for Novice–Per frequency (Initiation Phase) in Mailing Lists
Figure 65. Process Map for Novice–Per Average Duration (Initiation Phase) in Mailing List
The process map depicted in Figures 64 and 65 represents a workflow for all the activities performed by the Novice during the first phase of the learning processes. As hinted earlier, the numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 64 while they represent the average duration of occurrence in Figure 65 for each path. We focus on these two metrics in order to provide a quick overview of the model however our emphasis would be to look more at the traces of these activities.

In Figure 65, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply denotes that performing the activity occurs instantly within any process, while between activities there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Identifying an Expert and Formulating a question) to provide these details as an illustration in Figure 66.

![Figure 66. Highest average time explained](image)

From Figure 66, we gather that the path from “Identify Expert” to “Formulate Question” has an absolute frequency of 1427, this is the total number of times these two activities link up in this way repeating about 753 times at an average duration of 61.8 hours when the longest occurrence took about 29.8 weeks (after a Novice identified a potential Expert, they only came back to post a question about 30 weeks later).

In the next set of figures the idea is to represent the inherent statistical information about Novices with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge seekers, Novices).
Chapter 6: Process Maps for Initiation phase: Empirical Results

Figure 67. Events over time for Novice Model

Figure 68. Details about Activities for Novice Model

Figure 69. Details about Participants for Novice Model

Figure 70. Details about Learning States for Novice Model
In Figure 67, we note that Novice for this period of time we considered was active in a total of 122838 events as indicated in the main model performing a total of 7 activities over an average period of 70.1 days in 560 processes. These activities as indicated in Figure 68 were executed on average 17548.29 times with the most executed activity occurring 39829 times just like in the main model. Christopher Ferris’s contribution of 45 activities can be noticed as it helps towards a cumulative 85.53% of the total number of activities. Additionally, Figure 69 indicate that participants have played the role of Novice about 2446 with most of their activities being the 5 activities of the observation state representing 99.54% of activities executed. Lastly, we can be sure that these details concern only Novices as shown in Figure 71.
Figure 72. Process map for Expert–Per frequency (Initiation Phase) in Mailing Lists
Figure 73. Process map for Expert–Per Average Duration (Initiation Phase) in Mailing List
The process map depicted in Figures 72 and 73 represents a workflow for all the activities performed by the Expert during the first phase of the learning processes. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 72, while they represent the average duration of occurrence in Figure 73.

In Figure 73, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply denotes that performing the activity occurs instantly within any process while between activities there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Commenting on a Post and Making Contact with a Novice) to provide these details as an illustration in Figure 74.

From Figure 74 we gather that this path has an absolute frequency of 12, this is the total number of times these two activities link up in this way repeating about 5 times at an average duration of 50.7 days when the longest occurrence took about 32.1 weeks.

In Figures 75 – 79, the idea is to represent the inherent statistical information about Experts with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge providers, Experts).
Figure 75. Events over time for Expert Model

Figure 76. Details about Activities for Expert Model

Figure 77. Details about Participants for Expert Model

Figure 78. Details about Learning States for Expert Model
This set of figures summarizes and give core performance indicators for the process map representing the Expert. In Figure 75, we note that Expert for this period of time we considered was active in a total of 440 events as indicated in the main model performing a total of 6 activities over an average period of 18.1 weeks in 30 processes. These activities as indicated in Figure 76 were executed on average 3.17 times with the most executed activity occurring 39 times. Andi Abes is considered for illustration purpose to show his contribution of 3 activities helping towards a cumulative 62.27% of the total number of activities. Additionally, Figure 77 indicates that participants have played the role of Expert about 139 times with activities in the first state representing 57.73%, while the second state takes up the remaining 42.27%. Lastly, we can be sure that these details concern only Experts as shown in the last figure in the set.

The Mailing archives dataset has proved largely to be an adequate environment to identify all activities pertaining to either making contacts or beginning of potential collaboration. It has proved extensively that in FLOSS, learning processes start at great scale. From a total of 54762 emails exchanged on 15 mailing lists, it was possible to identify 14 learning activities executed in 565 cases over 123401 events. The Novice performs more of these Initiation phase activities with 99, 54 % of all the activities, while the Expert accounts for only about 0.36% of these events. Nevertheless, we look at a different repository in order to get a different perspective.
6.3 Internet Relay Chats’ Messages

Unlike the Mailing archives dataset, this dataset is a database with only 3 tables. These tables contain details of chat messages between conversers; the channels people are registered to or engage into communication on as well as the details about the people. The channels correspond to the topic of discussions in the case of mailing archives. While people send emails based on a particular topic, here the chats are group on channels representing a topic of interest or discussion forum.

<table>
<thead>
<tr>
<th>Tables</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo.channels</td>
<td>30</td>
</tr>
<tr>
<td>dbo.irclog</td>
<td>5603302</td>
</tr>
<tr>
<td>dbo.people_uperson</td>
<td>19247</td>
</tr>
</tbody>
</table>

This repository contains more than 5 million chat messages, exactly 5603302, exchanged between 19247 people on a combined total of 30 channels/forums. From these chats messages, we eliminated those that could not be linked to senders. The final dataset was made up of 2142690 chat messages that we analyzed. Just like emails, these chat messages were exchanged over a period of 3 and half years. The first message was sent on 2010-07-28 at 05:09:11 and the last message we considered was exchanged on 2014-04-09 at exactly 18:07:19. Furthermore, it is fit to mention the significant difference in the message length between the Mailing archives and Internet Relay Messages repositories. In this dataset, the average length was 60 characters; the longest chat message reached 502 characters while the shortest message was of single character length.

We mined this data accordingly and the resulting process maps are graphical representations of the occurrence of learning activities in these forums. Three process maps result from this endeavor in conformity with the same procedure followed on emails. The first process map represents the full learning process while the last two are representations of activities undertaken by the Novice and Expert respectively.

In order to fully grasp the processes as well as the underlying paths, we explore the process maps using two different views according to the two sets of metrics described in Chapter 5. Moreover, these views are intertwined with four main attributes considered to describe the process maps.

It is crucial to remind that the main or full process map is presented with absolute frequency and total duration in order to give an overview picture of what and how processes have taken place on chat forums. Evidently, some participants would understandably play both Novices and Experts roles on different scales and at different times. However, as we break down the models according to the Novice and Expert roles, we consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.
Figure 80. Process Map for Initiation Phase – Per frequency of occurrence in Internet Relay Chats
Chapter 6: Process Maps for Initiation phase: Empirical Results

Figure 81. Process Map for Initiation Phase – Per Duration of Activities in Internet Relay Chats
In Figures 80 and 81, the represented process map portrays how learning activities occur while people exchange messages in discussion forums. Once again, the numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 80, while they represent the total duration of occurrence in Figure 81.

Key information represented in our models includes the discussion topic represented by distinct channels, the message itself, the people involved in exchanging these emails, the resulting activities and classification of where they fall in our defined learning curve. This simply means the activities can be clustered according to their time of occurrence, the senders as well as the roles played and states of learning process as message exchange takes place.

Additional details regarding the numerical measures and occurrence flow of activities such as events over time, active cases during this given period of time, case variants, the number of events per case as well as case duration could be plotted as needed. However, for simplicity and effectiveness, we represent only major statistical details that are most representative of the presence, impact and occurrence of learning activities in FLOSS over the chosen period of time. We explain this information in Figures 82 – 85.

Figure 82. Events over time

A representation of the time frame for the occurrence of our process events is given in Figure 82. The right hand side of the figure gives some details with regards to input data used in the plot. The log timeline on the horizontal axis represents the total timeframe covered by our dataset (from earliest to latest timestamp observed). The figure shows the level of activity in the process by plotting the number of performed activities in the process on the vertical axis.

The process map representing the Initiation Phase of the learning process started on the 28th of July 2010 and ended on the 9th of April 2014. We note that, during this time, a total of 605,965 events were generated. An event represents a tuple made up of the case (in this context, the discussion topic), the chat message senders as well as the relevant learning activities. With about 28 cases, a total of 14 activities are executed with an average time per case of 35.7 weeks while the median duration is of 22.4 weeks.

In Figure 83, we give a representation of the case duration metric that shows the throughput time of the process from the very beginning (start of first activity) to the very end (completion of last activity). For example, in Figure 83 below one can see that there are a number of short-running cases (as seen, 2 cases out of the 28 considered in this analysis last up to 14 days and 19 hours) and then, towards the right, a smaller number of very long-running cases (up to 3 years or more) which can be explained by the volatile nature of discussions in FLOSS. In this instance, it appears that people send a lot of chat messages over a selected period of time and as the conversations evolve, they are prone to add more and contribute to the discussions making the process instance lasting up to 3 years and 260 days for the last one represented in Figure 83.
The next critical attribute we now consider is activity. Learning activities can help determine the flow and occurrence of the learning process as well as the participants. In Figure 84, details about activities can be observed.

The statistics about activities depict performance metrics with regard to the role and impact factor for each single activity in the process. Five specific indicators shown at the right hand side of the chart include the number of activities (14), the minimal frequency (1332) which speaks of how often the least frequent activity has occurred, the mean frequency which represents the average occurrence of an activity (43283.21), the maximal frequency (131888) which depicts how often the most frequent activity has occurred and finally the frequency std. deviation (49177.48) which is simply the standard deviation for the frequency of activities.

In comparison with the results obtained from emails dataset, we notice a number of resemblances in the occurrence of activities except for minor differences in the frequency of occurrence. These differences can be explained by the discrepancies in message sizes and numbers in both repositories. Looking at Figure 84, we can observe that the activity “Comment Post” by the Novice is executed 21.76% of the process occurrence. This activity is triggered by a set of key phrases, including “does not work”, “not executing”, “this does not work for me”, “do not know what is wrong with my code”, “here is
my code”, “in short my problem”, “step by step”, “details provided”, “works as follows”, “I want it to”, “expect it to”, “my question is like this”, “what I mean”. These key phrases identify messages (emails) sent by the Novice expressing a number of feelings, opinions or needs. These can range from expressing the fact that whatever is being discussed did not work for him, or give clarification concerning his initial request by rephrasing the question (concern) or even showing the error to indicate the kind of problem he/she is facing. The following activity, “Post Message” is closely related to the third one (Identify Expert) because it is its direct resultant.

The activity “Identify Expert” occurs 20.55% of the process occurrence. This activity is triggered by a set of key phrases, including “How did you do this” or “I saw your code” or “I need your help” or “this does not work for me” or “-1” or “is this possible to do this” or “can this be done?” or “very helpful” or “very well” or “can you explain”. These key phrases defined messages that assume that the person whose message is being responded to has a certain level knowledge of the question or a particular subject. It would appear that if they posted a response, or some step by step indications on how to solve a particular problem, a lot of people would comment on it, some with the need to ask for clarification, others with sense of acquiescence, some with the need to make contact for further exchange. We indicate that “Post Message” is a resultant of this activity based on the assumption that when people have posted messages expressing the need to get help or having identified someone more knowledgeable about their problems, there is a need to distinctively document that through “Post Message” activity.

The rest of activities are proportionally and relatively less because, we assume at this point, that interaction has occurred. After a Novice has posted a message, there has been a reaction and hence the rest of the process unfolds. Also to note here is the appearance of the activity “Participate in Discussions” which is an activity that the participants engage into beside the ones we have predefined and looked for in the dataset. As we semantically searched the data, the analysis is that when the emails meet our predefined criteria, they can be classified according to identified activities, in the contrary they should be labeled as a generic activity called “Participate in Discussions”. This label represents all other activities that possibly take place in FLOSS during this first phase of learning process other than the one identified in our specification.

In the next two subsections we focus on each learning participant’s role, Novice and Expert. But before then, it is imperative that for consistency we provide global indicators of our discovered process map with regards to three other attributes namely learning participants, states as well as roles in Figure 85 – 87.

Similarly to activities, overall performance facts about participants in the process are provided in Figure 85. It should be noted that during this 3 and half year time frame, a total of 2454 FLOSS members took part in this process (in its first phase) where they sent at least 1 message, a maximum of 4010 activities per participant at an average of 50.29 activities for a 183.05 standard deviation.
While we cannot list all the participants, the Pareto graph in the figure exemplifies how some details about a single participant can be obtained. The displayed yellow line shows the cumulative, relative sum (how much out of 100%) for the values below. In this case, jc_smith has executed about 40 activities and when added to the previous sum, it helps reach 95.26% of all executed activities. This in essence shows the progression in terms of each participant’s contributions for the complete process map.

In Figure 86, it can be noted that three main states were registered in the model namely the Observation, ContactEstablishment and Participation. While the first two are recorded in our specification, the last state is the classification for all activities and participants that do not fall in our initial categorization as highlighted in Section 6.3. As with the activities, when it is found that a message’s content specifies a different activity other than the one indicated in the specification, we conclude that the resulting activity is part of the “Inactive” state as the performer in this context is excluded from the learning process.

In Figure 86, just like in the previous analysis of the Mailing archives repository, the biggest share in terms of activities goes to the observation state (517558 in total) with an average frequency of 201988.33 while the standard deviation is 273291.75. The figure depicts that the second state has about 44672 activities getting the cumulative sum to just 92.78%.
Finally, we can note how participants in quest for knowledge justifiably claim the majority of activities with a total of 524332 amounting to 86.53% of all executed activities at this point in contrast with Experts who intervene at a lower rate of 6.25% with 37898 activities, slightly behind activities related to doing something other than exchanging knowledge with 43735 activities.

We give more lights about what can possibly justify this discrepancy in the next two subsections as we unpack activities flow for Novices and Experts respectively. As indicated earlier, we shall consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.
Figure 88. Process Map for Novice–Per frequency (Initiation Phase) in Internet Relay Chats
Figure 89. Process Map for Novice–Per Average Duration (Initiation Phase) in Internet Relay Chats
Figures 88 and 89 provide a pictorial representation for the workflow of all the activities performed by the Novice during the first phase of the learning processes. As hinted earlier, the numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 88, while they represent the average duration of occurrence in Figure 89 for each path. We focus on these two metrics in order to provide a quick overview of the model however our emphasis would be to look more at the traces of these activities.

In Figure 88, one can notice that the thickness of the edges is the same. We considered case frequency to identify how many cases were involved in constructing the workflow. In this instance, all 28 cases are concerned. Unlike the emailing lists where some processes might have been incomplete in some instances, the degree to which people engage in sending instant messages and the motivation therein can justify the presence of the workflow in almost every channel.

In Figure 89, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply denotes that performing the activity occurs instantly within any process while between activities there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Identifying an Expert and Formulating a question) to provide these details as an illustration in Figure 90.

From this figure, we gather that the path from “Identify Expert” to “Formulate Question” has an absolute frequency of 31112, this is the total number of times these two activities link up in this way as a result of about 10874 repetitions at an average duration of 43.8 minutes for a total duration of 31.1 months.

In the next set of figures the idea is to represent the inherent statistical information about Novices with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge seekers, Novices).
Figure 91. Events over time for Novice Model

Figure 92. Details about Activities for Novice Model

Figure 93. Details about Participants for Novice Model
Figures 91 - 95 depict indications in terms of core performance for the process map representing the Novice. In Figure 91, we note that Novice for this period of time we considered was active in a total of 524332 events as indicated in the main model performing a total of 7 activities over an average period of 35.7 weeks in 20 processes. These activities, as indicated in Figure 92, were executed on average 74904.57 times with the most executed activity occurring 131888 times as previously indicated in the main model. Marcus’s contribution of 9 activities can be noticed as it helps towards a cumulative 98.37% of the total number of activities. Additionally, Figure 93 indicates that participants have played the role of Novice about 7431 times. Most of their activities are the 5 activities of the observation state representing 94.23% of activities executed. Lastly, we can be sure that these details concern only Novices as shown in Figure 95.
Figure 96. Process map for Expert–Per frequency (Initiation Phase) in Internet Relay Chats
Figure 97. Process map for Expert–Per Average Duration (Initiation Phase) in Internet Relay Chats
We can view the representation of the workflow for all the activities performed by the Expert during the first phase of the learning processes in Figures 96 and 97. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 96, while they represent the average duration of occurrence in Figure 97 for each path.

In Figure 96, one can notice that the thickness of the edges is almost the same. We considered case frequency to identify how many cases were involved in constructing the workflow. This figure is indicative of the flow occurring in almost all 28 cases with the lowest being 23. Unlike the Mailing archives where some processes might have been incomplete in some instances, the degree to which people engage in sending instant messages in IRC and the motivation therein can justify the presence of the workflow in almost every channel.

In Figure 97, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply denotes that performing the activity occurs instantly within any process while between activities there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Commenting on a Post and Reading Source Code) to provide these details as an illustration in Figure 98.

From Figure 98, we gather that this path has an absolute frequency of 745. This is the total number of times there is a link between CommentPost and ReadSourceCode. Also, we can note that the maximum repetitions is about 300 times at an average duration of 7.9 hours while the longest occurrence took about 19 days.
The inherent statistical information about Experts with regards to a number of attributes including events per case, the participants, learning states as well as role is provided I Figures 99 – 103.

Figure 99. Events over time for Expert Model

Figure 100. Details about Activities for Expert Model

Figure 101. Details about Participants for Expert Model
Considering the core performance indicators for the Expert’s process map, we note in Figure 99 that Expert was active in a total of 37898 events performing a total of 6 activities over an average period of 35.3 weeks in 28 processes. These activities as indicated in Figure 100 were executed on average 6316.33 times with the most executed activity occurring 19506 times. Sebastian is considered for illustrative purpose to depict his contribution of 3 activities helping towards a cumulative 96.12% of the total number of activities. Additionally, Figure 101 indicates that participants have played the role of Expert about 2814 times with activities in the first state representing 62.01%, while the second state takes up the remaining 37.99%. Lastly, we can be sure that these details concern only Experts as shown in Figure 103.

Considering this description, one can observe practically the same trend on IRC messages as on Mailing archives. Learning activities do occur at a significant rate during messages exchange on these forums and the results from mining IRC messages provide insights on the starting activities that trigger these processes. While the dataset boasts over 5 million messages, only 2142690 of these were good enough for our analysis. It has emerged that 14 of the learning activities could be identified through 605965 events across 28 cases where the Novice, accounts for the biggest share of traffic during these exchange. The Novice executes the activities about 86, 53% of the total number compared to the Expert who operates the related activities for 7.22%.
6.4 Conclusion

In this chapter, the empirical results expressed through process maps give significant indications to the existence of learning activities in FLOSS communities. We produced these maps in 3 different ways in order to allow for a more simplified approach to represent the occurrence of learning activities in FLOSS. Starting with the full Initiation phase, we depict the process map for FLOSS members and learning participants regardless of their roles (Novice or Expert). We then break it down and clearly depict process maps for each participant in the respective role. Looking at the quantitative properties, we note that learning activities do occur at a significant rate during messages exchange on both Mailing archives and IRC messages. The slight differences between the two datasets can be highlighted in two ways.

Firstly, we note that Mailing archives appear to produce more events linked to learning activities compared to IRC messages given the proportionality in terms of dataset sizes. On the one hand, with only 54762 email messages in Mailing archives, the total number of corresponding events is more than double the number of emails. On the other hand, the tendency leads in the opposite direction on IRC messages. While we analyzed millions of messages exchanged on IRC, the total corresponding number of events is 605965 representing almost 1/4 times of the number of messages exchanged.

A possible explanation might be that people might consider chat forums more like social environments than serious platforms for serious engagement. Lastly, the involvement of Experts is more on IRC than it is on Mailing archives with 7.22% and 0.36% of Expert involvement respectively on IRC forums and Mailing lists. This can probably be justified by the differences in length of messages sent on these two datasets. The average length for message sent is 3261 characters for an email compared to 60 characters for a chat message. It might be that it is easier to respond to a shorter message on IRC forums than a long thread of emails on Mailing archives. Nevertheless, the results from these two repositories speak volume on the existence of learning activities in FLOSS. This is critical as it provides for a practical way to satisfy all hypotheses made in this work in this regard.

In the next two chapters, 7 and 8, we replicate this exact same approach to provide details and describe the corresponding process maps for the second and third phases of the learning process.
CHAPTER 7:
PROCESS MAPS FOR PROGRESSION PHASE:
EMPIRICAL RESULTS

7.1 Introduction
In this second phase of the learning process, activities are undertaken across three main steps (states) for both participants after establishing contact or reacting to one another’s request. These are Revert, Post and Apply. While the focus here is not to model these states, we simply recall that this set of activities in this phase are meant to providing some level of feedback after being contacted, then providing the required help for Novice and implementing new knowledge through a set of new activities.

Just like with the first phase, here we also consider the same two repositories: Mailing archives and IRCs. We have generated the relevant logs from these datasets and have produced full models with absolute frequency and total duration. The main purpose thereof is to provide the reader with a more globalized view of what and how processes take place in FLOSS at this stage, especially when participants can play both the Novice and Expert roles. However, as we break down the models in Novice and Expert roles, we consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.

7.2 Mailing archives’ Messages
The workflow net for learning activities occurrence during this phase can be noted as depicted in Figures 104 and 105. As a point of emphasis, the numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 104, while they represent the total duration of occurrence in the second Figure 105. Process instances are demarcated as a result of the use of topic of discussion as our case as we retained the topic of emails (or thread), the message itself, and the people involved in exchanging these emails, the resulting activities and classification of where they fall in our defined learning curve.

Additional details regarding the numerical measures and occurrence flow of activities such as events over time, active cases during this given period of time, case variants, the number of events per case as well as case duration could be plotted as needed. However, for simplicity and effectiveness, we represent only major statistical details that are most representative of the presence, impact and occurrence of learning activities in FLOSS over the chosen period of time. We explain this information in subsequent figures starting with Figure 106.
Figure 104. Process map for Progression Phase – Per frequency of occurrence in Mailing Lists
Figure 105. Process map for Progression Phase – Per Duration of Activities in Mailing Lists
Figure 106. Events over time

In Figure 106, a representation of the time frame for the occurrence of our process events is given. The right hand side of the figure gives some details with regards to input data used in the plot. The log timeline on the horizontal axis represents the total timeframe covered by our dataset (from earliest to latest timestamp observed). The figure shows the level of activity in the process by plotting the number of performed activities in the process on the vertical axis.

The process map representing the Progression Phase of the learning process started on the 11th of November 2010 and ended on the 6th of May 2014. During this time, we note that a total of 117014 events were generated. An event represents a tuple made up of the case (in this context, the discussion topic), the email senders as well as the relevant learning activities. With about 579 cases, a total of 21 activities are executed with an average time per case of 72.3 days while the median duration is of 63.1 days.

In the next figure, we give a representation of the case duration metric that shows the throughput time of the process from the very beginning (start of first activity) to the very end (completion of last activity). For example, in Figure 107 below one can see that there are a number of short-running cases (as seen, 342 cases last up to 12 days and 17 hours) and then, towards the right, a smaller number of very long-running cases (up to 3 years or more) almost the same trend as the initiation phase on the emails. This depicts that the same momentum is maintained across the board with regard to actively participating in knowledge exchange for both the Novice and the Expert.

Figure 107. Case duration

Moreover, the fact that the majority of cases (about 69%) take up to 12 days and 17 hours is indicative of how much interactions people create and engage in and eventually culminating into learning processes. For a period of over 3 years, in the majority of cases, people participated in discussions and reacted to information exchange relative fast either during the initiation or the current progression phase.
The next critical metric we now consider is activity. Learning activities are pivotal in determining the flow and occurrence of the learning process as well as the participants. In the following figure, details about activities can be observed.

![Activity Details for Process Activities](image)

**Figure 108. Details for Process Activities**

The statistics about activities depict performance metrics with regard to the role and impact factor for each single activity in the process. On Mailing archives in this phase, one can critically retain the number of activities (21), the minimal frequency (3) which speaks of how often the least frequent activity has occurred, the mean frequency which represents the average occurrence of an activity (5572.1), the maximal frequency (39408) which depicts how often the most frequent activity has occurred and finally the frequency std. deviation (11,916.82) which is simply the standard deviation for the frequency of activities.

Looking at Figure 108, we can observe that in this phase, it seems that the Novice gets busier in gradually reporting bugs. Hence, the activity “Report Bugs” by the Novice is executed 33.68% of this phase process occurrence. This activity is triggered by a set of key phrases, including “seriously this needs to be updated” or “now obsolete” or “not compatible” or “send me your code” or “what is your problem?” or “this works for me” or “Did it work before”, “I think it should work”. These key phrases identify messages (emails) sent by the Novice expressing a number of feelings, opinions or offer to help as well as a level of feedback like with “this works for me” or “Did it work before”, “I think it should work”. While the first key phrase/sentence expresses frustration (“seriously this needs to be updated”), it also shows that something is not running properly or there is a bug.

The following activity, “Provide Feedback” is indicative of the willingness for the Novice to show how far the help he/she got was put in practice. This is expressed in all the messages or even portions of the emails sent with key phrases like “Still not working” or “but same error” or “It comes up with the following error” or “Many thanks” or “I am using” or “is it a bug?” or “Is there a workaround?” or “in case” or “suppose that” or “is this the same” or “how come” or “what is happening?” or “What am I doing wrong”. And the evidence shows that this activity occurs around 30.68% of the times considered at this
phrase. The rest of the activities complete the process and depict providing feedbacks on different levels, asking more questions, commenting on messages etc. as shown in Figure 108.

Also to note here is the presence of the activity “Participate in Discussions” which is an activity that the participants engage into beside the ones we have predefined and looked for in the dataset. As we semantically searched the data, the analysis is that when the emails meet our predefined criteria, they can be classified according to identified activities, in the contrary they should be labeled as a generic activity called “Participate in Discussions”. This label represents all other activities that possibly take place in FLOSS during this second phase of learning process other than the one identified in our specification.

In the next two subsections we focus on each learning participant’s role, Novice and Expert. But before then, it is imperative that for consistency we provide global indicators of our discovered process map with regards to three other attributes namely learning participants, states as well as roles in the next three figures.

In Figure 109, just like with activities, overall performance facts about participants in the process are provided. It should be noted that during this 3 and half year time frame, a total of 2533 FLOSS members took part in this process (in its second phase) where they executed at least 1 activity, executed a maximum of 4725 activities per participant at an average of 46.2 activities for a 189.33 standard deviation.

While we cannot list all the participants, the Pareto graph in the figure exemplifies how some details about a single participant can be obtained. The displayed yellow line shows the cumulative, relative sum (how much out of 100%) for the values below. In this case, Andrew Hutchings has executed about 26 activities and when added to the previous sum, it helps reach 90.47% of all executed activities. This in essence shows the progression in terms of each participant contributions for the complete process map.

In Figures 110 and 111, we show how the model has been built in the context of learning states as well as roles (Novice, Expert, and Inactive). In Figure 110 specifically, it can be noted that four states were registered in the model namely the Apply, Revert, Post and Participation. While the first three are recorded in our specification, the last state is the classification for all activities and participants that do not fall in our initial categorization. As with the activities, when it is found that a message’s content specifies a different activity other than the one indicated in the specification, we conclude that the resulting activity is part of the “Inactive” state as the performer in this context is excluded from the learning process.
Figure 110. Details about States

In Figure 110, the lion share in terms of activities goes to the Apply state (43632 in total) with an average frequency of 29253.5 while the standard deviation is 19430.16. The figure depicts that the second state has about 36452 activities getting the cumulative sum to just 68.44%, while Post is trailing the second busiest state with 31.08% of the total activities undertaken during this phase.

Figure 111. Details about Roles

Finally, we can note how participants in quest for knowledge justifiably claim the majority of activities with a total of 112207 activities amounting to 95.89% of all executed activities at this point in contrast with Experts who intervene at a lower rate of 3.63% with 4248 activities, ahead of people doing something other than exchanging knowledge with 559 activities.

We give more lights about what can possibly justify this discrepancy in the next two subsections as we unpack activities flow for Novices and Experts respectively. As indicated earlier, we shall consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.
Figure 112. Process Map for Novice–Per frequency (Progression Phase) in Mailing Lists
Figure 113. Process Map for Novice–Per Average Duration (Progression Phase) in Mailing List
Consistently with our representation approach, Figures 112 and 113 depict a workflow for all the activities performed by the Novice during the second phase of the learning processes through a process map. As hinted earlier, the numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 112, while they represent the average duration of occurrence in the second figure for each path. We focus on these two metrics in order to provide a quick overview of the model however our emphasis would be to look more at the traces of these activities.

In Figure 113, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply demonstrates that performing the activity occurs instantly within any process while from one activity to another there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Commenting on Code and Posting Questions) to provide these details as an illustration in Figure 114.

From this figure, we gather that the path from “Comment On Code” to “Post Questions” has an absolute frequency of 392, this is the total number of times these two activities link up in this way repeating about 138 times at an average duration of 21.2 hours when the longest occurrence took about 13.3 weeks.

In the next set of figures the idea is to represent the inherent statistical information about Novices with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge seekers, Novices).
Chapter 7: Process Maps for Progression Phase: Empirical Results

Figure 115. Events over time for Novice Model

Figure 116. Details about Activities for Novice Model

Figure 117. Details about Participants for Novice Model

Figure 118. Details about Learning States for Novice Model
This set of figures summarizes and gives core performance indicators for the process map representing the Novice. In the first figure of the set (Figure 115), we note that the Novice for this period of time we considered was active in a total of 112207 events as indicated in the main model performing a total of 8 activities over an average period of 72.5 days in 568 processes. These activities as indicated in Figure 116 were executed on average 14025.87 times with the most executed activity occurring 39408 times just like in the main model. Loic Dachary’s contribution of 11 activities can be noticed as it helps towards a cumulative 95.53% of the total number of activities. Additionally, Figure 117 indicates that participants have played the role of Novice about 2499 times with most of their activities being the activities of the Apply state representing 37.59% of activities executed. Lastly, we can be sure that these details concern only Novices as shown in the last figure in the set.

In the following subsections, we describe the process discovery endeavor related to the Experts. Similar to the Novice, we learn highlight activities as they occur for Experts during this phase on this data set.
Figure 120. Process Map for Expert–Per frequency (Progression Phase) in Mailing Lists
Figure 121. Process Map for Expert–Per Average Duration (Progression Phase) in Mailing List
We note the relevant process map in Figures 120 and 121 depicting a workflow for all the activities performed by the Expert during the second phase of the learning processes. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 120, while they represent the average duration of occurrence in the second figure for each path.

In Figure 121, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply demonstrates that performing the activity occurs instantly within any process while from one activity to another there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Running Source Code and Reporting Bugs) to provide these details as an illustration in Figure 122.

From this figure, we gather that this path has an absolute frequency of 1. In this process instance, it has only occurred once that the Expert would run the code and indicate on in the email sent that potential bugs through “Report Bugs” activity. The total duration is of 34.8 days.

In the next set of figures the idea is to represent the inherent statistical information about Experts with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge providers, Experts).
Chapter 7: Process Maps for Progression Phase: Empirical Results

Figure 123. Events over time for Expert Model

Figure 124. Details about Activities for Expert Model

Figure 125. Details about Participants for Expert Model
This set of figures summarizes and give core performance indicators for the process map representing the Expert. In Figure 123, we note that Expert for this period of time we considered was active in a total of 4248 events as indicated in the main model performing a total of 12 activities over an average period of 67.6 days in 163 processes. These activities as indicated in Figure 124 were executed on average 354 times with the most executed activity occurring 1285 times. Chris Jones is considered for illustration purpose to show his contribution of 8 activities helping towards a cumulative 74.32% of the total number of activities. Additionally, Figure 125 indicates that participants have played the role of Expert about 695 times with activities in the first state representing 58.66%, while the second and third states take up respectively 28.95% and 12.38%. Lastly, we can be sure that these details concern only Experts as shown in the last figure in the set.

Furthermore, a very important indicator at this stage is that although Experts have performed activities less frequently than Novices, they actually executed more activities than the Novices. On a total of 21 activities, the Experts performed 12 of those and in majority sending feedback, answering questions, posting more clarification questions etc. We shall examine the total contribution and this variation trend in participations in the next phase as well. However, before that we examine the same patterns as they are observed on IRC.
7.3 Internet Relays Chat Messages

The workflow net for learning activities occurrence during this phase can be noted as depicted in Figures 129 and 130. As a point of emphasis, the numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 129 while they represent the total duration of occurrence in Figure 130. Process instances are demarcated thanks to the use of topic of discussion as our case as we retained the topic of emails (or thread), the message itself, and the people involved in exchanging these emails, the resulting activities and classification of where they fall in our defined learning curve.

Additional details regarding the numerical measures and occurrence flow of activities such as events over time, active cases during this given period of time, case variants, the number of events per case as well as case duration could be plotted as needed. However, for simplicity and effectiveness, we represent only major statistical details that are most representative of the presence, impact and occurrence of learning activities in FLOSS over the chosen period of time.

![Events over time](image)

Figure 128. Events over time

In Figure 128, a representation of the time frame for the occurrence of our process events is given. The right hand side of the figure gives some details with regards to input data used in the plot. The log timeline on the horizontal axis represents the total timeframe covered by our dataset (from earliest to latest timestamp observed). The figure shows the level of activity in the process by plotting the number of performed activities in the process on the vertical axis.
Figure 129. Process Map for Progression Phase – Per frequency of occurrence in IRC messages
Figure 130. Process Map for Progression Phase – Per Duration of Activities in IRC messages
Different to the mailing lists archives dataset, the process map representing the Progression Phase of the learning process on these chat messages started on the 28th of July 2010 and ended on the 9th of April 2014. During this time, we note that a total of 739578 events were generated. An event represents a tuple made up of the case (in this context, the channel/forum name), the message senders as well as the relevant learning activities. With about 28 cases representing each separate channel grouped according to forum discussion topic, a total of 21 activities are executed with an average time per case of 35.7 weeks while the median duration is of 22.4 weeks.

In Figure 131, we give a representation of the case duration metric that shows the throughput time of the process from the very beginning (start of first activity) to the very end (completion of last activity). For example, in Figure 131 below one can see that for this environment, the duration of cases varies according to the number of events and activity frequency. The figure depicts 2 cases out of the total 28 last up to 14 days and 8 hours while others would go from 54 days to 3 years and 260 days. This depicts that the same momentum is maintained across the board with regard to actively participating in knowledge exchange for both the Novice and the Expert.

Moreover, the fact that the majority of cases (just 6 cases) take up to 121 days and 15 hours and this is indicative of how much interactions people create and engage in and eventually culminating into learning processes. The willingness and loyalty to participate in the discussions can be seen in the growing number of activities executed as well as the duration of the respective cases. For a period of over 3 years, in the majority of cases, people participated in discussions and reacted to information exchange relative fast either during the initiation or the current progression phase. The next critical metric we now consider is activity. Learning activities are pivotal in determining the flow and occurrence of the learning process as well as the participants. In the following figure, details about activities can be observed.
Figure 132. Details for Process Activities

The statistics about activities depict performance metrics with regard to the role and impact factor for each single activity in the process. On chat messages in this phase, one can critically retain the number of activities (21), the minimal frequency (9) which speaks of how often the least frequent activity has occurred. This dataset records three times the minimal frequency as observed on Mailing archives (3). The average occurrence of an activity at this point for this dataset is also largely superior to that of activities on emails amounting to 35218 while the maximal frequency (125391) which depicts how often the most frequent activity has occurred and finally the frequency std. deviation (43405.94) which is simply the standard deviation for the frequency of activities.

Looking at the figure above, we can observe that in this phase, it seems that the Novice gets busier in gradually reporting bugs just like in the Mailing archives. Hence, the activity “Report Bugs” by the Novice is executed 125391 times (16.95%) of this phase process occurrence. This activity is triggered by a set of key phrases, including “seriously this needs to be updated” or “now obsolete” or “not compatible” or “send me your code” or “what is your problem?” or “this works for me” or “Did it work before”, “I think it should work”. These key phrases identify messages (emails) sent by the Novice expressing a number of feelings, opinions or offer to help as well as a level of feedback like with “this works for me” or “Did it work before”, “I think it should work”. While the first key phrase/sentence expresses frustration (“seriously this needs to be updated”), it also shows that something is not running properly or there is a bug.

The following activity, “Participate in Discussions” identifies all other activities one engaged in outside of the predefined main learning activities. Nonetheless, the third largest number of frequencies is for “Provide Feedback” activity. This is indicative of the willingness for the Novice to show how far the help he/she got was put in practice. This is expressed in all the messages or even portions of the emails sent with key phrases like “Still not working” or “but same error” or “It comes up with the following error” or “Many thanks” or “I am using” or “is it a bug?” or “Is there a workaround?” or “in case” or “suppose that” or “is this the same” or “how come” or “what is happening?” or “What am I doing wrong”. And the evidence shows that this activity was executed around 115232 times (15.58%) of the times considered at
this phrase. The rest of the activities complete the process and depict providing feedbacks on different levels, asking more questions, commenting on messages etc. as shown in Figure 132.

In the next two subsections we focus on each learning participant’s role, Novice and Expert. But before then, it is imperative that for consistency we provide global indicators of our discovered process map with regards to three other attributes namely learning participants, states as well as roles in the next three figures.

In Figure 133, just like with activities, overall performance facts about participants in the process are provided. It should be noted that during this 3 and half year time frame, a total of 8520 FLOSS members took part in this process (in its second phase) where they executed at least 1 activity, executed a maximum of 57231 activities per participant at an average of 86.8 activities for a 935.72 standard deviation.

While we cannot list all the participants, the Pareto graph in the figure exemplifies how some details about a single participant can be obtained. The displayed yellow line shows the cumulative, relative sum (how much out of 100%) for the values below. Also it is worth noting that on these chat messages, users are identified with their nicknames rather than their real names. In this case, JerryKwan has executed about 26 activities and when added to the previous sum, it helps reach 99.22% of all executed activities. This in essence shows the progression in terms of each participant contributions for the complete process map.

In the last two figures we show how the model has been built in the context of learning states as well as roles (Novice, Expert, and Inactive). In Figure 134 specifically, it can be noted that four states were registered in the model namely the Apply, Revert, Post and Participation. While the first three are recorded in our specification, the last state is the classification for all activities and participants that do not fall in our initial categorization. As with the activities, when it is found that a message’s content specifies a different activity other than the one indicated in the specification, we conclude that the resulting activity is part of the “Inactive” state as the performer in this context is excluded from the learning process.
In Figure 134, the lion share in terms of activities, similarly to the mailing dataset, goes to the Apply state (337979 in total) with an average frequency of 184894.5 while the standard deviation is 103613.13. The figure depicts that the second state has about 158926 activities getting the cumulative sum to just 67.19%, while Post is trailing the second busiest state with 16.82% of the total activities undertaken during this phase.

Finally, we can note how participants in quest for knowledge justifiably claim the majority of activities with a total of 339811 activities amounting to 45.95% of all executed activities at this point in contrast with Experts who intervene at a lower rate of 38.06% with 281478 activities, ahead of people doing something other than exchanging knowledge with 118289 activities.

We give more lights about what can possibly justify this discrepancy in the next two subsections as we unpack activities flow for Novices and Experts respectively. As indicated earlier, we shall consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.
Figure 136. Process Map for Novice–Per frequency (Progression Phase) in IRC messages
Figure 137. Process Map for Novice–Per Average Duration (Progression Phase) in IRC messages
The process map depicted in Figures 136 and 137 represents a workflow for all the activities performed by the Novice during the second phase of the learning processes. As hinted earlier, the numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 136 while they represent the average duration of occurrence in the second figure for each path. We focus on these two metrics in order to provide a quick overview of the model however our emphasis would be to look more at the traces of these activities.

In Figure 137, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply demonstrates that performing the activity occurs instantly within any process while from one activity to another there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Commenting on Code and Posting Questions) to provide these details as an illustration in Figure 138.

From this figure, we gather that the path from “Comment On Code” to “Post Questions” has an absolute frequency of 1220, this is the total number of times these two activities link up in this way repeating about 497 times at an average duration of 49.8 minutes when the longest occurrence took about 3.1 days.

In the next set of figures the idea is to represent the inherent statistical information about Novices with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge seekers, Novices).
Chapter 7: Process Maps for Progression Phase: Empirical Results

Figure 139. Events over time for Novice Model

Figure 140. Details about Activities for Novice Model

Figure 141. Details about Participants for Novice Model
This set of figures summarizes and gives core performance indicators for the process map representing the Novice. Figure 139 shows that the Novice was active in a total of 339811 events as indicated in the main model, performing a total of 8 activities over an average period of 37.5 weeks in 28 processes. These activities as indicated in Figure 140 were executed on average 42476.38 times with the most executed activity occurring 125391 times just like in the main model. Ryan Lane’s contribution of 6 activities can be noticed as it helps towards a cumulative 98.81% of the total number of activities. Additionally, Figure 141 indicates that participants have played the role of Novice about 7581 times with most of their activities being the activities of the Apply state representing 48.19% of activities executed. Lastly, we can be sure that these details concern only Novices as shown in the last figure in the set.

Next, we look at the processes discovered in the same data set for the Expert.
Figure 144. Process Map for Expert–Per frequency (Progression Phase) in IRC messages
Figure 145. Process Map for Expert–Per Average Duration (Progression Phase) in IRC messages
The process map depicted in Figures 144 and 145 represents a workflow for all the activities performed by the Expert during the second phase of the learning process on the IRC dataset. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 144 while they represent the average duration of occurrence in the second figure for each path.

In Figure 145, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply demonstrates that performing the activity occurs instantly within any process while from one activity to another there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Posting Questions and Reporting Bugs) to provide these details as an illustration in Figure 146.

From this figure, we gather that this path has an absolute frequency of 1893. In this process instance, it has occurred 1893 times that the Expert would run post questions for either clarification or to point the Novice to a learning artifact before pointing out potential bugs through “Report Bugs” activity. The total duration is of 44 days.

In the next set of figures the idea is to represent the inherent statistical information about Experts with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge providers, Experts).
Chapter 7: Process Maps for Progression Phase: Empirical Results

Figure 147. Events over time for Expert Model

Figure 148. Details about Activities for Expert Model

Figure 149. Details about Participants for Expert Model
This set of figures summarizes and gives core performance indicators for the process map representing the Expert. In Figure 147 we note that Expert for this period of time we considered was active in a total of 281478 events as indicated in the main model performing a total of 12 activities over an average period of 37.5 weeks in 28 processes. These activities as indicated in Figure 148 were executed on average 23456.5 times with the most executed activity occurring 86969 times. Mike Dawson is considered for illustration purpose to show his contribution of 10 activities helping towards a cumulative 97.69 % of the total number of activities. Additionally, Figure 149 indicates that participants have played the role of Expert about 6137 times with activities in the first state representing 61.9%, while the second and third states take up respectively 22.96% and 15.14 %.Lastly, we can be sure that these details concern only Experts as shown in the last figure in the set.

Furthermore, it is important to note that almost the same trend can be observed on both datasets with regard to the increase in the involvement of the Expert. Performing 12 activities out of a total of 21 activities, the Expert exemplifies the commitment and willingness to assist and help as we discussed in Chapter 1. A major difference in comparison with the Mailing archives dataset is that the ordering for the states with most activities has shifted. While in the Mailing archives the most frequent state was Post, here we notice that the Expert executes more activities during Apply.
7. 4 Conclusion

Both the Mailing archives and IRC messages datasets have been mined again to trace learning activities as they occur during this second phase. The evidence produced from this mining experiment solidifies the finding in terms of the existence of learning processes in FLOSS as well as the scale at which they occur. While the first phase proved that the Novice is more involved in the start of the learning process, here the involvement of the Expert can be seen to be significantly increasing. The same trend in terms of the number of events in comparison with the overall size of both datasets highlighted in the first phase can also be verified in this phase. In Chapter 6, we noticed that there were more events on Mailing archives than there were on IRC messages. However, in this second phase, we can notice a shift in the level of interest from the Expert’s side with the continuous commitment by the Novice to progress through the full cycle of the learning process.

Primarily with Mailing archives, the total number of events produced amounts to 117014 with the Novice still performing the most activities at 95.89 % while the Expert increased from 0.36 to 3.63 % of the total learning activities. Secondly, on IRC messages, 739578 events are produced over 28 cases where the Novice contributes 45.95 % of the total number of events, while the Expert performs up to 38.06%. This explains the willingness for Experts to intervene more during chat forums than on mailing lists.

Furthermore, although the intensity in terms of number of activities sent is still low vis-à-vis with the Novice, the Expert performs more activities than the Novice. This amounts to 12 activities out of a total of 21 activities. During this time, the Expert gets involved and performs activities that range from sending feedback, answering questions, posting more clarification questions to providing step by step guidance.
CHAPTER 8:
PROCESS MAPS FOR MATURATION PHASE:
EMPIRICAL RESULTS

8.1 Introduction
This is the last chapter in terms of describing process discovery from our datasets. We therefore put
conclude with the last phase of the learning process, maturation. In this third and last phase, activities are
undertaken across five main steps (states) for both participants to indicate the extent to which at this level,
more advanced skills are transferred across different platforms. These states include Analyze, Commit,
Develop, Revert and Review.

A very distinguishing trait to note regarding this phase is that, all participants at this level are considered
pairs with different skillset that they are sharing among them. More importantly, as someone’s skillset has
matured, we identify a Novice as anybody asking a question or requesting some information about
something they were not acquainted with prior, while the Expert is any respondent to these requests. In
order to trace these advanced skills, we look at activities performed by participants who tend towards the
last layer of contributors, the core developers as seen in Figure 2 (from Chapter 1). Such activities can be
identified by looking at repositories that store data about developing, creating code, examining and
reviewing the code, identifying and fixing possible bugs. Therefore, we consider three repositories as
Openstack has availed: Source Code, Bug reports and Reviews.

In Source Code, the emphasis is about getting the feel about how participants react to pieces of code as
they are committed, how far they personally contribute in terms of code submission, reviewing code that
has been submitted as well as providing feedback with regards to any comment or question asked in the
process. The Bug reports dataset contains detailed data about bug reporting. Specifically, information
related to bugs, the person it is assigned to, the person who reported the bug, the bugged file as well as
accompanying details and comments. In the last dataset, it is reported general comments regarding
submitted files and related reports of reviews and how far these reviews have been attended to.

Having generated the relevant Event Logs from these datasets, we make use of the same approach as in
the previous phases to represent the process maps starting from full models with absolute frequency and
total duration in order to provide the reader with a more globalized view of what and how processes take
place in FLOSS at this stage, then we break down the maps according to the Novice and Expert roles
considering only case frequency for the first array of metrics as well as the average duration in order to
provide insights within the bounds of our experiments.

8.2 Bug reports
This database is made up of 10 tables that we considered. These tables contain data about the attachments
(file, source code, and bugged file), the changes that were made and the corresponding versions, the
comments or messages posted by the participants in this regard, the issues (bugs) themselves as they are
described, the issues watchers, the people and supported trackers as well as the duration as shown in the
table below.
Table 12. Bug reports dataset Description

<table>
<thead>
<tr>
<th>Tables</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo.attachments</td>
<td>3763</td>
</tr>
<tr>
<td>dbo.changes</td>
<td>779641</td>
</tr>
<tr>
<td>dbo.comments</td>
<td>147615</td>
</tr>
<tr>
<td>dbo.issues</td>
<td>38580</td>
</tr>
<tr>
<td>dbo.issues_ext_launchpad</td>
<td>38459</td>
</tr>
<tr>
<td>dbo.issues_log_launchpad</td>
<td>818221</td>
</tr>
<tr>
<td>dbo.related_to</td>
<td>827</td>
</tr>
<tr>
<td>dbo.trackers</td>
<td>70</td>
</tr>
<tr>
<td>dbo.people</td>
<td>5368</td>
</tr>
<tr>
<td>dbo.people_upeople</td>
<td>5368</td>
</tr>
</tbody>
</table>

This repository contains exactly 3763 files that are reported to be reviewed and fixed. This is achieved by 5368 people. The bugs (issues) reported amount to initial 38580 while after reviews only 38459 are reported from the initial total and the general statistic (log) including related data is 818221. These bugs were reported during a period of time spanning from 2010 to 2014. The first issue recorded or the very first bug reported was at 20:52:34 on the 13th of July 2010 while the last bug included in this analysis was sent at exactly at 14:20:27 on the 6th of May 2014. Furthermore, the length of the messages considered is of an average of 437 characters length, the longest description/report was of 65535 characters (pretty much like the longest email sent) and the shortest message yields a single character length.

We analyzed this dataset and produced the process maps representing the occurrence of learning activities during this process of bug reporting and this output consists of a set of process maps. The first Process map represents the full learning process while the last two are representations of activities undertaken by the Novice and Expert respectively. Just like in the last two phases, in order to fully grasp the processes as well as the underlying paths, three main views are available to present the process maps. These include the map (graphical workflow), the statistics as well as cases. We consider the first two to provide a more efficient analysis of our data as the details we require from the last metric are absorbed in the first two.

Additionally, each process map can be represented with two different set of metrics on the basis of which the flow of events is explained. These metrics include frequency and performance. The main objective of the frequency metrics is the depiction of how often certain parts of the processes have been executed in this last phase. Therefore, we have produced full models with absolute frequency and total duration as these are meant to give an overview picture of what and how processes take place in FLOSS with some participants being both Novices and Experts at different times. However, as we break down the models in Novice and Expert roles, we consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.
Figure 152. Process Map for Maturation Phase – Per frequency of occurrence On Bug reports
Figure 153. Process Map for Maturation Phase – Per Duration of Activities On Bug reports
The resulting model depicted in Figures 152 and 153 represents a map of activities as they occur during this last phase of the learning processes. In a nutshell, it pens out a workflow representation of how learning activities are executed. Some important components can be noticed in the model. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 152, while they represent the total duration of occurrence in the second figure.

As already noted in this thesis, this process map can be represented with various degrees of annotations as defined by the needs of the investigation. In order to identify these advanced skills, we retained the issue (bug) number, the message accompanying every submissions, the people involved in submitting and commenting on these bugs, the resulting activities and classification of where they fall in our defined learning curve.

Additional details regarding the numerical measures and occurrence flow of activities such as events over time, active cases during this given period of time, case variants, the number of events per case as well as case duration could be plotted as needed. However, for simplicity and effectiveness, we represent only major statistical details that are most representative of the presence, impact and occurrence of learning activities in FLOSS over the chosen period of time. We explain this information in the following successive figures.

![Figure 154. Events over time](image)

In Figure 154, a representation of the time frame for the occurrence of our process events is given. The right hand side of the figure gives some details with regards to input data used in the plot. The log timeline on the horizontal axis represents the total timeframe covered by our dataset (from earliest to latest timestamp observed). The figure shows the level of activity in the process by plotting the number of performed activities in the process on the vertical axis.

For this period of time, we note that a total of 956649 events were generated. An event represents a tuple made up of the case (in this context, the issue number), the submitter, as well as the relevant learning activities. With about 31327 cases, a total of 36 activities are executed with an average time per case of 30.6 days while the median duration is of 6.3 days.

Figure 155 depicts the case duration metric that shows the throughput time of the process from the very beginning (start of first activity) to the very end (completion of last activity) during this phase. One can see that there are a number of short-running cases (as seen, 20248 cases last up to 9 days and 4 hours) and then, towards the right, a smaller number of very long-running cases (the last case lasting 2 years and 199 days) which can be explained by the volatile nature of discussions in FLOSS.
Additionally, the fact that the majority of cases (about 64.6%) take up to 9 days and 4 hours as the maximal time is indicative, once again, of how much interactions people create and engage in and eventually culminate into learning processes. For a period of over 3 years, in the majority of cases, people participated in discussions and reacted to information exchange relatively fast.

The next critical metric we now consider is activity. Learning activities can help determine the flow and occurrence of the learning process as well as the participants. In the following figure, details about activities can be observed.

![Figure 156. Details for Process Activities](image)
The statistics about activities depict performance metrics with regard to the role and impact factor for each single activity in the process. Five specific indicators shown at the right hand side of the chart include the number of activities (36), the minimal frequency (66) which speaks of how often the least frequent activity has occurred, the mean frequency which represents the average occurrence of an activity (26573.58), the maximal frequency (90748) which depicts how often the most frequent activity has occurred and finally the frequency std. deviation (32704.09) which is simply the standard deviation for the frequency of activities.

Looking at Figure 156, we can observe that the activity “Analyze Source Code” by the Expert is executed 9.49% of the process occurrence. This activity is triggered by a set of key phrases, including “problem with your code”, “error at line”, “correct it”, “try this”, “remove line”, “fix this line”, “try again”, “troubleshooting”, “rerun it”, “it looks good to me”, “it should work”, “remove lines”, “change data type”, “syntax error”. These key phrases identify comments on code that anybody had in response to a request for help. This is directly related to the second activity “Run Source code” which simply implies that when a source code has been analyzed, it is highly probable that it was first run.

The third most executed activity is by the Novice. This activity is “Review Source Code”. It can be identified if the comments posted can be found to be determined by these key phrases: “It is a bug”, “file not working properly”, “bug fixed in next version”, “it is not”, “it should work”, “I checked”, “error at line”, “fix that”, “try again”, “troubleshooting”, “rerun it”, “build failed”. These comments are given as a feedback and express the fact that the respondents did some reviewing of the source code as concerned in this instance.

The rest of activities are proportionally and relatively less because, we assume at this point, it did not require long to explain why a bug has been reported and that the latter has been described or attended to satisfactorily. Also to note here is the apparition of the activity “Participate in Discussions” which is an activity that the participants engage into beside the ones we have predefined and looked for in the dataset. This label represents all other activities that possibly take place in FLOSS during this last phase of learning process other than the one identified in our specification. We shall discuss more about this in the next chapter.

In the next two subsections we focus on each learning participant’s role, Novice and Expert. But before then, it is imperative that for consistency we provide global indicators of our discovered process map with regards to three other attributes namely learning participants, states as well as roles in the next three figures.

Figure 157 depicts the overall performance facts about participants in the process are provided. It should be noted that during this 3 and half year time frame, a total of 3156 FLOSS members took part in this process (in its last phase) where they executed at least 1 activity, a maximum of 571355 activities per participant at an average of 303.12 activities for a 10191.86 standard deviation.
Figure 157. Details about Participants

While we cannot list all the participants, the Pareto graph in the figure exemplifies how some details about a single participant can be obtained. The displayed yellow line shows the cumulative, relative sum (how much out of 100%) for the values below. In this case, Tyler North has executed about 81 activities and when added to the previous sum, it helps reach 99.05% of all executed activities. This in essence shows the progression in terms of each participant contributions for the complete process map.

In the last two figures we show how the model has been built in the context of learning states as well as roles (Novice, Expert, and Inactive). In figure x specifically, it can be noted that 6 main states were registered in the model namely the Develop, Review, Commit, Revert, Analyze and Participation. While the first five states are recorded in our specification, the last state is the classification for all activities and participants that do not fall in our initial categorization. As with the activities, when it is found that a message’s content specifies a different activity other than the one indicated in the specification, we conclude that the resulting activity is part of the “Inactive” state as the performer in this context is excluded from the learning process.

Figure 158. Details about States

In Figure 158, the lion share in terms of activities goes to the Develop state. This state is defined by global key phrase such as “upload”, “update”, “patch”, “module”, “created”, “fixed”, “code”, “changed”. These indicate a range of actions that relate to Software and application development. The idea is that if a participant refers to uploading, creating, fixing, coding or key terms like patch, they are dealing with develop activities. Hence, in this state, 363773 activities in total were executed constituting about 38.03% of the total number of activities. The figure also depicts that the second state (Review) has about 359645 activities, the third state with the largest activities is Commit with 90156 followed by Revert with 52045.
getting the cumulative sum to just 90.51% while Analyze carries about 52026 activities slightly above the last state called participation with just about 38804 activities.

![Figure 159. Details about Roles](image)

Finally, we can note how participants in quest for knowledge justifiably claim the majority of activities with a total of 470879 activities amounting to 49.22% of all executed activities at this point in contrast with Experts who intervene at a lower rate of 46.72% with 446966 activities, ahead of people doing something other than exchanging knowledge with 38804 activities.

We give more lights about what can possibly justify this discrepancy in the next two subsections as we unpack activities flow for Novices and Experts respectively. As indicated earlier, we shall consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.
Figure 160. Process Map for Novice–Per frequency (Maturation Phase) on Bug Reports
Figure 161. Process Map for Novice–Per Average Duration (Maturation Phase) on Bug Reports
One can notice the process map representing a workflow for all the activities performed by the Novice during this crucial last phase of the learning process as depicted in Figures 160 and 161. As hinted earlier, the numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 160, while they represent the average duration of occurrence in the second figure for each path. We focus on these two metrics in order to provide a quick overview of the model however our emphasis would be to look more at the traces of these activities.

Figure 161, the model is displayed according to the average time spent within and between activities. Two indicators can be identified, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply demonstrates that performing the activity occurs instantly within any process while from one activity to another there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Reviewing Posts by the Novice to Modifying a Source Code) to provide these details as an illustration in Figure 162.

From this figure, we gather that the path from “Review Posts” to “Modify Source Code” has an absolute frequency of 277, this is the total number of times these two activities link up in this way repeating about 2 times at an average duration of 16.1 days when the longest occurrence took about 20.8 months.

In the next set of figures the idea is to represent the inherent statistical information about Novices with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge seekers, Novices).
Chapter 8: Process Maps for Maturation Phase: Empirical Results

Figure 163. Events over time for Novice Model

Figure 164. Details about Activities for Novice Model

Figure 165. Details about Participants for Novice Model
This set of figures summarizes and give core performance indicators for the process map representing the Novice. In Figure 163, we note that Novice for this period of time we considered was active in a total of 470879 events as indicated in the main model performing a total of 17 activities over an average period of 28.5 days in 30289 processes. These activities as indicated in Figure 164 were executed on average 27698.76 times with the most executed activity occurring 88606 times as indicated in the main model. Cole Robinson’s contribution of 25 activities can be noticed as it helps towards a cumulative 99.37% of the total number of activities. Additionally, Figure 165 indicates that participants have played the role of Novice about 3027 times with most of their activities being those of the Review state representing 37.95% of activities executed. Lastly, we can be sure that these details concern only Novices as shown in the last figure in the set.
Figure 168. Process Map for Expert–Per frequency (Maturation Phase) on Bug Reports
Figure 169. Process Map for Expert–Per Average Duration (Maturation Phase) on Bug Reports
Chapter 8: Process Maps for Maturation Phase: Empirical Results

The process map depicted in Figures 168 and 169 represents a workflow for all the activities performed by the Expert during the last phase of the learning processes. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 168 while they represent the average duration of occurrence in the second figure for each path.

In Figure 169 the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply demonstrates that performing the activity occurs instantly within any process while from one activity to another there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Providing Feedback and Analyzing Source Code) to provide these details as an illustration in Figure 170.

![Figure 170. Highest average time explained](image)

From this figure, we gather that this path has an absolute frequency of 8671, this is the total number of times these two activities link up in this way repeating about 11 times at an average duration of 15.9 days when the longest occurrence took about 27.5 months.

In the next set of figures the idea is to represent the inherent statistical information about Experts with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge providers, Experts).
Chapter 8: Process Maps for Maturation Phase: Empirical Results

Figure 171. Events over time for Expert Model

Figure 172. Details about Activities for Expert Model

Figure 173. Details about Participants for Expert Model
This set of figures summarizes and give core performance indicators for the process map representing the Expert. In Figure 171 we note that Expert for this period of time we considered was active in a total of 446966 events as indicated in the main model performing a total of 18 activities over an average period of 28.1 days in 30037 processes. These activities as indicated in Figure 172 were executed on average 24831.44 times with the most executed activity occurring 90748 times. Ryan Lane is considered for illustration purpose to show his contribution of 30 activities helping towards a cumulative 93.93% of the total number of activities. Additionally, Figure 173 indicates that participants have played the role of Expert about 2883 times with activities in the first state representing 41.59 %, while the second state takes up 40.52% of activities with the last State (revert) only occurring 3327 times. Lastly, we can be sure that these details concern only Experts as shown in the last figure in the set.
8.3 Reviews dataset

In this reviews dataset, one can find 7 tables as availed by Openstack. These tables contain data about changes reviewed and the corresponding versions, the comments or messages posted by the participants in this regard, the issues (bugs) themselves as they are described, the committers, the people and supported trackers as well as the duration as shown in Table 13.

<table>
<thead>
<tr>
<th>Tables</th>
<th>Records</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo.changes</td>
<td>314550</td>
<td></td>
</tr>
<tr>
<td>dbo.comments</td>
<td>581265</td>
<td></td>
</tr>
<tr>
<td>dbo.issues</td>
<td>30472</td>
<td></td>
</tr>
<tr>
<td>dbo.issues_ext_gerrit</td>
<td>30472</td>
<td></td>
</tr>
<tr>
<td>dbo.issues_log_gerrit</td>
<td>345022</td>
<td></td>
</tr>
<tr>
<td>dbo.people</td>
<td>2101</td>
<td></td>
</tr>
<tr>
<td>dbo.people_upeople</td>
<td>2100</td>
<td></td>
</tr>
</tbody>
</table>

This repository contains reviews about 314550 changes as performed by 5368 people. The reviews recorded amount to initial 30472 as consolidated to the same amount after verification, and the general statistic (log) including related data is 345022. These changes were affected during a period of time spanning from 2011 to 2014. Specifically, the first issue recorded reported was at 22:25:47 on the 31st of August 2011 while the last issue included in this analysis was sent at exactly at 11:14:26 on the 28th of April 2014. Furthermore, the length of the messages considered is of an average of 352 characters length, the longest review was of 10766 characters and the shortest message yields 15 characters of length.

As a result of mining this dataset, we produced the process maps representing the occurrence of learning activities from this data set and this output consists of process maps. The first process map represents the full learning process while the last two are representations of activities undertaken by the Novice and Expert respectively. Just like in the last two phases, in order to fully grasp the processes as well as the underlying paths, three main views are available to present the process maps. These include the map (graphical workflow), the statistics as well as cases. We consider the first two to provide a more efficient analysis of our data as the details we require from the last metric are absorbed in the first two.
Figure 176. Process map for Maturation Phase – Per frequency of occurrence On Reviews
Figure 177. Process map for Maturation Phase – Per Duration of Activities On Reviews
The resulting model depicted in Figures 176 and 177 represents a map of activities as they occur during this last phase of the learning processes. In a nutshell, it pens out a workflow representation of how learning activities are executed. Some important components can be noticed in the model. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 176, while they represent the total duration of occurrence in the second figure.

As already noted in this thesis, this process map can be represented with various degrees of annotations as defined by the needs of the investigation. In order to identify these advanced skills, we retained the issue (patch or file reviewed) number, the message accompanying every submissions, the people involved in submitting and reviewing on these issues, the resulting activities and classification of where they fall in our defined learning curve.

Additional details regarding the numerical measures and occurrence flow of activities such as events over time, active cases during this given period of time, case variants, the number of events per case as well as case duration could be plotted as needed. However, for simplicity and effectiveness, we represent only major statistical details that are most representative of the presence, impact and occurrence of learning activities in FLOSS over the chosen period of time. We explain this information in the following successive figures.

In Figure 178, a representation of the time frame for the occurrence of our process events is given. The right hand side of the figure gives some details with regards to input data used in the plot. The log timeline on the horizontal axis represents the total timeframe covered by our dataset (from earliest to latest timestamp observed). The figure shows the level of activity in the process by plotting the number of performed activities in the process on the vertical axis.

For this period of time, we note that a total of 12225111 events were generated. A significant difference in the number of events when taken in context with the first dataset we analyzed. Perhaps an interesting trait here is the emphasis that every time there is a file committed, or any type of commits, a critical number of people are willing to run it and review it and under any circumstance this process would always yield a certain level of knowledge transfer as outcome. As far as our dataset is concerned, about 28624 cases are in question. This is the magnitude of files or patches that were reviewed. The total number of activities is 36, executed with an average time per case of 10.7 days while the median duration is of 6 days.

In the next figure, we give a representation of the case duration metric that shows the throughput time of the process from the very beginning (start of first activity) to the very end (completion of last activity)
during this phase. For example, in Figure x below one can see that there are a number of short-running cases (as seen, 20248 cases last up to 5 days and 7 hours) and then, towards the right, a smaller number of very long-running cases (the last case lasting 1 year and 173 days) which can be explained by the nature of reviews and certainly the type of file being reviewed.

![Figure 179. Case duration](image)

Additionally, the fact that the majority of cases (about 64.3%) take up to 5 days and 7 hours as the maximal time is indicative, once again, of how much interactions people create and engage in and eventually culminate into learning processes. This is almost on par with the level of interactions on the bug reporting dataset where the majority of the events (64.6%) also took about up to 7 days for full execution. For a period of over 3 years, in the majority of cases, people participated in discussions and reacted to information exchange relatively fast.

The next critical metric we now consider is activity. Learning activities can help determine the flow and occurrence of the learning process as well as the participants. In the following figure, details about activities can be observed.
The statistics about activities depict performance metrics with regard to the role and impact factor for each single activity in the process. Five specific indicators shown at the right hand side of the chart include the number of activities (36), the minimal frequency (46) which speaks of how often the least frequent activity has occurred, the mean frequency which represents the average occurrence of an activity (33958.64), the maximal frequency (458093) which depicts how often the most frequent activity has occurred and finally the frequency std. deviation (78718.45) which is simply the standard deviation for the frequency of activities.

Interestingly, there is an indication that at this point in this environment, people do engage in a number of other activities through “Participate in Discussions”. However, the next most popular activity is, just like in the previous dataset, the activity “Analyze Source Code” by the Expert which is executed 9.9% of the process occurrence (also slightly on par with Bug Reporting dataset). This activity is triggered by a set of key phrases, including “problem with your code”, “error at line”, “correct it”, “try this”, “remove line”, “fix this line”, “try again”, “troubleshooting”, “rerun it”, “it looks good to me”, “it should work”, “remove lines”, “change data type”, “syntax error”. These key phrases identify comments on code that anybody had in response to a request for help. This is directly related to the third activity “Comment On Code” which simply implies that when a source code has been analyzed, this is shown through the comment put on it.

The fourth activity, “Analyze Thread Progression” simply speaks of a participant intervening on a discussion already started and providing his/her contribution. This is triggered by key phrases like “Looks good”, “looks correct”, “no”, “-1”, “+1”, “inline comment”, “don’t think so”, “build succeeded”. These comments are given as a feedback and express the fact that the respondents did some reviewing of the
issue being discussed as concerned in this instance. The rest of activities are proportionally and relatively less because, we assume at this point, it did not require long to explain why a bug has been reported and that the latter has been described or attended to satisfactorily.

In the next two subsections we focus on each learning participant’s role, Novice and Expert. But before then, it is imperative that for consistency we provide global indicators of our discovered process map with regards to three other attributes namely learning participants, states as well as roles in the next three figures.

Figure 181 depicts the overall performance facts about participants in the process are provided. It should be noted that during this 3 and half year time frame, a total of 1642 FLOSS members took part in this process (in its last phase) where they executed at least 1 activity, a maximum of 185280 activities per participant at an average of 744.53 activities for a 6231.94 standard deviation.

While we cannot list all the participants, the Pareto graph in the figure exemplifies how some details about a single participant can be obtained. The displayed yellow line shows the cumulative, relative sum (how much out of 100%) for the values below. In this case, Victor Howard has executed about 54 activities and when added to the previous sum, it helps reach 99.26% of all executed activities. This in essence shows the progression in terms of each participant contributions for the complete process map.

Figures 182 and 183 show how the model has been built in the context of learning states as well as roles (Novice, Expert, and Inactive). In Figure 182 specifically, it can be noted that 6 main states were registered in the model namely the Develop, Review, Commit, Revert, Analyze and Participation. While the first five states are recorded in our specification, the last state is the classification for all activities and participants that do not fall in our initial categorization. As with the activities, when it is found that a message’s content specifies a different activity other than the one indicated in the specification, we conclude that the resulting activity is part of the “Inactive” state as the performer in this context is excluded from the learning process.
In Figure 182, the lion share in terms of activities goes to Participation state where a number of activities other than the identified ones take place. Our learning process’s first traced state is the Develop state which comes right after the Participation State. This state is defined by global key phrase such as “upload”, “update”, “patch”, “module”, “created”, “fixed”, code, and “changed”. These indicate a range of actions that relate to Software and application development.

The idea is that if a participant refers to uploading, creating, fixing, coding or key terms like patch, they are dealing with develop activities. Hence, in this state, 358547 activities in total were executed constituting about 29.33 % of the total number of activities. The figure also depicts that the third state (Review) has about 150178 activities, the fourth state with the largest activities is Analyze with 106817 activities while Commit and Revert have respectively 104958 and 43918 activities.

Finally, we can note how participants with a solid knowledge base or advanced skills justifiably claim the majority of activities with a total of 493355 activities amounting to 40.36% of all executed activities. At the same time, participants also engage in more other activities than asking or giving answers to questions with about 37.47 % of the total number of activities while about only 22.17 % of activities of the total number are executed by participants in their capacity as knowledge seekers.

We give more lights about what can possibly justify this discrepancy in the next two subsections as we unpack activities flow for Novices and Experts respectively. As indicated earlier, we shall consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.
Figure 184. Process map for Novice–Per frequency (Maturation Phase) on Reviews
The process map depicted in Figures 184 and 185 represents a workflow for all the activities performed by the Novice during this crucial last phase of the learning processes. As hinted earlier, the numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 184 while they represent the average duration of occurrence in the second figure for each path. We focus on these two metrics in order to provide a quick overview of the model however our emphasis would be to look more at the traces of these activities.

In Figure 185, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply demonstrates that performing the activity occurs instantly within any process.
while from one activity to another there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Reporting Bugs by the Novice to Analyzing Thread progression) to provide these details as an illustration in figure x below.

From this figure, we gather that the path from “Report Bugs” to “Analyze Thread Progression” has an absolute frequency of 3077, this is the total number of times these two activities link up in this way repeating about 11 times at an average duration of 9.1 days when the longest occurrence took about 87.9 days.

In the next set of figures the idea is to represent the inherent statistical information about Novices with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge seekers, Novices).

![Figure 186. Highest average time explained](image)

![Figure 187. Events over time for Novice Model](image)
Figure 188. Details about Activities for Novice Model

Figure 189. Details about Participants for Novice Model

Figure 190. Details about Learning States for Novice Model
This set of figures summarizes and give core performance indicators for the process map representing the Novice. In Figure 187, we note that Novice for this period of time we considered was active in a total of 271063 events as indicated in the main model performing a total of 17 activities over an average period of 8.7 days in 27980 processes. These activities as indicated in Figure 188 were executed on average 15944.88 times with the most executed activity occurring 39765 times as indicated in the main model. Dustin J. Mitchell’s contribution of 10 activities can be noticed as it helps towards a cumulative 99.59% of the total number of activities. Additionally, Figure 188 indicates that participants have played the role of Novice about 1274 times with most of their activities being those of the Develop state representing 42.22% of activities executed. Lastly, we can be sure that these details concern only Novices as shown in the last figure in the set.
Figure 192. Process map for Expert–Per frequency (Maturation Phase) on Reviews
Figure 193. Process map for Expert–Per Average Duration (Maturation Phase) on Reviews
The process map depicted in Figures 192 and 193 represents a workflow for all the activities performed by the Expert during the last phase of the learning processes. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 192 while they represent the average duration of occurrence in the second figure for each path.

In Figure 192 the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply demonstrates that performing the activity occurs instantly within any process while from one activity to another there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Commenting on Code and Reviewing Posts) to provide these details as an illustration in Figure 194.

From this figure, we gather that this path has an absolute frequency of 27603, this is the total number of times these two activities link up in this way repeating about 42 times at an average duration of 6.1 minutes when the longest occurrence took about 36.9 days.

In the next set of figures the idea is to represent the inherent statistical information about Experts with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge providers, Experts).
Figure 195. Events over time for Expert Model

Figure 196. Details about Activities for Expert Model

Figure 197. Details about Participants for Expert Model
This set of figures summarizes and give core performance indicators for the process map representing the Expert. Figure 195 shows that Expert for this period of time we considered, was active in a total of 493355 events as indicated in the main model performing a total of 18 activities over an average period of 9.3 days in 28087 processes. These activities as indicated in Figure 196 were executed on average 27408.61 times with the most executed activity occurring 121030 times. Alexandre Levine is considered for illustration purpose to show his contribution of 7 activities helping towards a cumulative 83.58% of the total number of activities. Additionally, Figure 197 indicates that participants have played the role of Expert about 1267 times with activities in the first state representing 49.48%, while the second state takes up 18.36% of activities with the last State (Revert) only occurring 3931 times. Lastly, we can be sure that these details concern only Experts as shown in the last figure in the set.
8.4 Source Code dataset

Openstack has gathered all there is to know directly pertaining to commits, submitted codes as well as description and comments on the submitted code in this dataset. This database is made up of 13 tables that we have considered. These tables contain data about the actions undertaken by developers, the pieces of codes (branches) as they commit them, the commit lines, files, their copies, the domains appropriate for these commits, the tags and their revisions, the projects as well as the people involved in these actions as shown in Table 14.

<table>
<thead>
<tr>
<th>Tables</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbo.actions</td>
<td>425744</td>
</tr>
<tr>
<td>dbo.branches</td>
<td>17</td>
</tr>
<tr>
<td>dbo.commits_lines</td>
<td>131556</td>
</tr>
<tr>
<td>dbo.compannies/projects</td>
<td>210</td>
</tr>
<tr>
<td>dbo.domains</td>
<td>1258</td>
</tr>
<tr>
<td>dbo.files</td>
<td>93584</td>
</tr>
<tr>
<td>dbo.file_links</td>
<td>114077</td>
</tr>
<tr>
<td>dbo.file_types</td>
<td>75441</td>
</tr>
<tr>
<td>dbo.tags</td>
<td>324</td>
</tr>
<tr>
<td>dbo.tag_revisions</td>
<td>889</td>
</tr>
<tr>
<td>Dbo.comments</td>
<td>131556</td>
</tr>
<tr>
<td>dbo.people</td>
<td>2677</td>
</tr>
<tr>
<td>dbo.people_uppeople</td>
<td>2677</td>
</tr>
</tbody>
</table>

This repository contains exactly 93584 source code files that are reported to be committed. This is achieved by 2677 people. These people performed a number of actions and these among to 425744 on about 210 projects as reported. The files submitted are of 75441 types. This piece of information was particularly used to identify whether a file committed was documentation, user manual or patch or even any general source file. These bugs were reported during a period of time spanning from 2010 to 2014. About 131556 messages can be identified as they pertain to the committed files. The first message recorded was at 23:05:26 on the 27th of May 2010 while the last message included in this analysis was sent at exactly at 12:27:48 on the 6th of May 2014. Furthermore, the length of the messages considered is of an average of 182 characters length, the longest description/report was of 8628 characters and the shortest message yields a single character length.

We analyzed this dataset following the same pattern as the previous repositories. Thus, we have produced full models with absolute frequency and total duration as these are meant to give an overview picture of what and how processes take place in FLOSS with some participants being both Novices and Experts at different times. However, as we break down the models in Novice and Expert roles, we consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.
Figure 200. Process Map for Maturation Phase – Per frequency of occurrence On Source Code
Figure 201. Process map for Maturation Phase – Per Duration of Activities On Source Code
The resulting model depicted in Figures 200 and 201, represents a map of activities as they occur during this last phase of the learning processes. In a nutshell, it pens out a workflow representation of how learning activities are executed. Some important components can be noticed in the model. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 200, while they represent the total duration of occurrence in the second figure.

As already noted in this thesis, this process map can be represented with various degrees of annotations as defined by the needs of the investigation. In order to identify these advanced skills, we retained the issue (committed code file, documentation) number, the message accompanying every submissions, the people involved in submitting and commenting on these files, the resulting activities and classification of where they fall in our defined learning curve.

Additional details regarding the numerical measures and occurrence flow of activities such as events over time, active cases during this given period of time, case variants, the number of events per case as well as case duration could be plotted as needed. However, for simplicity and effectiveness, we represent only major statistical details that are most representative of the presence, impact and occurrence of learning activities in FLOSS over the chosen period of time.

In Figure 202, a representation of the time frame for the occurrence of our process events is given. The right hand side of the figure gives some details with regards to input data used in the plot. The log timeline on the horizontal axis represents the total timeframe covered by our dataset (from earliest to latest timestamp observed). The figure shows the level of activity in the process by plotting the number of performed activities in the process on the vertical axis.

For this period of time, we note that a total of 1097281 events were generated. An event represents a tuple made up of the case (in this context, the issue number), the submitter, as well as the relevant learning activities. With about 100 cases, a total of 37 activities are executed with an average time per case of 21.5 months while the median duration is of 20.6 months.

In Figure 203, we give a representation of the case duration metric that shows the throughput time of the process from the very beginning (start of first activity) to the very end (completion of last activity) during this phase. Unlike the previous two repositories where most cases last shorter, here there is a distinguishing variation in terms of duration. The illustration in Figure x below shows how the duration fluctuates with the highest number of case, 6 in total taking up to a year and 5 days to fully execute, while the first two processes last up to 66 days and 2 hours) and then, towards the right, a smaller number of very long-running cases.
The next critical metric we now consider is activity. Learning activities can help determine the flow and occurrence of the learning process as well as the participants. Details about activities can be observed in Figure 204.

The statistics about activities depict performance metrics with regard to the role and impact factor for each single activity in the process. Five specific indicators shown at the right hand side of the chart include the number of activities (37), the minimal frequency (57) which speaks of how often the least frequent activity has occurred, the mean frequency which represents the average occurrence of an activity (29656.24), the maximal frequency (204488) which depicts how often the most frequent activity has
occurred and finally the frequency std. deviation (43668.91) which is simply the standard deviation for the frequency of activities.

An interesting trait in this repository is the difference in the number of activities as compared to the previous two. It is to be noted that for the last two datasets, there was no existence of activities pertaining to submitting documentation while it is possible here. This dataset gives us a list of file types and documents submitted as well as the details of the submitters. Furthermore, looking at the figure above, we can justifiably observe that the activity “Submit Source Code” tops the list of the executed activities. This shows another important factor. As it turns out, the Novice as the learning process matures can commit source code quite significantly as he/she practices the acquired skills. This activity is executed 204488 times (18.64%) and can be identified twofold. Firstly, If type of commit (file type) = “code”, or “patch”. Secondly, it can also be triggered by key phrases like “I submitted code”, ”my code”, “I wrote these lines”, “my file not working”, ”I corrected the new version”, “corrected this module”, “patch working” while checking for fixed file, updates, or imported files from issues.

The second most executed activity is “Participate in Discussions” which encompasses all activities not concerned with learning processes. The third most executed activity is by the Expert. This activity is “Review Code”. It can be identified if the comments posted can be found to be determined by these key phrases: “It is a bug”, “file not working properly”, “bug fixed in next version”, “it is not!”, ”it should work”, “I checked”, “error at line”, “fix that”, “try again”, “troubleshooting”, “rerun it”, “build failed”. These comments are given as a feedback and express the fact that the respondents did some reviewing of the source code as concerned in this instance. The rest of activities are proportionally and relatively less because, we assume at this point, it did not require long to explain why a bug has been reported and that the latter has been described or attended to satisfactorily.

In the next two subsections we focus on each learning participant’s role, Novice and Expert. But before then, it is imperative that for consistency we provide global indicators of our discovered process map with regards to three other attributes namely learning participants, states as well as roles in the next three figures. In Figure 205, we observe that the overall performance facts about participants in the process are provided. It should be noted that during this 3 and half year time frame, a total of 2237 FLOSS members took part in this process (in its last phase) where they executed at least 1 activity, a maximum of 372930 activities per participant at an average of 490.51 activities for a 8267.6 standard deviation.

![Figure 205. Details about Participants](image)

While we cannot list all the participants, the Pareto graph in the figure exemplifies how some details about a single participant can be obtained. The displayed yellow line shows the cumulative, relative sum (how much out of 100%) for the values below. In this case, Zed Shaw has executed about 72 activities...
and when added to the previous sum, it helps reach 98.99% of all executed activities. This in essence shows the progression in terms of each participant contributions for the complete process map.

In Figure 206, it can be noted that 6 main states were registered in the model namely the Develop, Review, Commit, Revert, Analyze and Participation. While the first five states are recorded in our specification, the last state is the classification for all activities and participants that do not fall in our initial categorization. As with the activities, when it is found that a message’s content specifies a different activity other than the one indicated in the specification, we conclude that the resulting activity is part of the “Inactive” state as the performer in this context is excluded from the learning process.

Furthermore, Figure 206 demonstrates that the majority of activities are executed during the Commit state. This state is defined by global key phrase such as “file”, “bug”, “patch”, and “module”. These indicate a range of actions that relate to committing a piece of code, patch or file. The idea is that if a participant refers to uploading a file, creating a file, fixing a bug, coding or key terms like patch or module, they are dealing with Commit activities. Hence, in this state, 382300 activities in total were executed constituting about 34.84% of the total number of activities.

Figure 206 also depicts that the second state Develop has about 283244 activities, the third state with the largest activities is Review with 218510 followed by Participation with 129186 activities, and Revert follows with 48199 activities getting the cumulative sum to just 96.73% while Analyze completes the set with just about 35839 activities.
Finally, we can note how participants in quest for knowledge justifiably claim the majority of activities with a total of 506814 activities amounting to 46.19% of all executed activities at this point in contrast with Experts who intervene at a lower rate of 42.04% with 461278 activities, ahead of people doing something other than exchanging knowledge with 129189 activities.

We give more lights about what can possibly justify this discrepancy in the next two subsections as we unpack activities flow for Novices and Experts respectively. As indicated earlier, we shall consider only case frequency for the first array of metrics as well as the average duration in order to provide insights within the bounds of our experiments.
Figure 208. Process map for Novice–Per frequency (Maturation Phase) on Source Code
Figure 209. Process map for Novice–Per Average Duration (Maturation Phase) on Source Code
The process map depicted in Figures 208 and 209 represents a workflow for all the activities performed by the Novice. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 208, while they represent the average duration of occurrence in the second figure for each path. We focus on these two metrics in order to provide a quick overview of the model however our emphasis would be to look more at the traces of these activities.

In Figure 209, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply demonstrates that performing the activity occurs instantly within any process while from one activity to another there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Reporting Bugs by the Novice to Writing Source Code) to provide these details as an illustration in Figure 210.

![Figure 210. Highest average time explained](image)

From this figure, we gather that the path from “Report Bugs” to “Write Source Code” has an absolute frequency of 197, this is the total number of times these two activities link up in this way repeating about 40 times at an average duration of 28.1 hours when the longest occurrence took about 60.1 days.

In the next set of figures the idea is to represent the inherent statistical information about Novices with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge seekers, Novices).
Chapter 8: Process Maps for Maturation Phase: Empirical Results

Figure 211. Events over time for Novice Model

Figure 212. Details about Activities for Novice Model

Figure 213. Details about Participants for Novice Model
This set of figures summarizes and give core performance indicators for the process map representing the Novice. Figure 211 shows that the Novice was active in a total of 506814 events as indicated in the main model performing a total of 18 activities over an average period of 21.2 months in 100 processes. These activities as indicated in Figure 212 were executed on average 28156.33 times with the most executed activity occurring 204488 times as indicated in the main model. Russell Sim’s contribution of 49 activities can be noticed as it helps towards a cumulative 98.72% of the total number of activities. Additionally, Figure 213 indicates that participants have played the role of Novice about 2214 times with most of their activities being those of the Commit state representing 41.65% of activities executed. Lastly, we can be sure that these details concern only Novices as shown in the last figure in the set.
Figure 216. Process map for Expert–Per frequency (Maturation Phase) on Source Code
Figure 217. Process map for Expert–Per Average Duration (Maturation Phase) on Source Code
We can see the Expert’s execution of activities in Figures 216 and 217. The numbers, the thickness of the arcs or edges, and the coloring in the model illustrate how frequent each activity or path has been performed on one hand in Figure 216 while they represent the average duration of occurrence in the second figure for each path.

In Figure 217, the model is displayed according to the average time spent within and between activities. Two indicators can be noticed, namely the label “instant” and a duration in a corresponding time unit. The indicator instant simply demonstrates that performing the activity occurs instantly within any process while from one activity to another there is certainly some time incurred. This average time is calculated taking into account the frequency and repetition metrics indicators in play. We pick a particular path (between Commenting On Code and Analyzing Source Code) to provide these details as an illustration in Figure 218 below.

From this figure, we gather that this path has an absolute frequency of 1297, this is the total number of times these two activities link up in this way repeating about 222 times at an average duration of 29 hours when the longest occurrence took about 25.3 weeks.

In the next set of figures the idea is to represent the inherent statistical information about Experts with regards to a number of attributes including events per case, the participants, learning states as well as role (to make sure here we only have data about participants when they are Knowledge providers, Experts).
Chapter 8: Process Maps for Maturation Phase: Empirical Results

Figure 219. Events over time for Expert Model

Figure 220. Details about Activities for Expert Model

Figure 221. Details about Participants for Expert Model
This set of figures summarizes and give core performance indicators for the process map representing the Expert. In Figure 219 we note that Expert for this period of time we considered, was active in a total of 461,278 events as indicated in the main model performing a total of 18 activities over an average period of 21.2 months in 100 processes. These activities as indicated in Figure 220 were executed on average 25,626.56 times with the most executed activity occurring 99,052 times. Jay Payne is considered for illustration purpose to show his contribution of 39 activities helping towards a cumulative 96.8% of the total number of activities. Additionally, Figure 221 indicates that 2,212 participants have played the role of Expert with activities in the first state representing 37.11%, while the second state takes up 32.95% of activities with the last State (Revert) only occurring 10,916 times. Lastly, we can be sure that these details concern only Experts as shown in the last figure in the set.

Figure 222. Details about Learning States for Expert Model

Figure 223. Evidence that the process map represents only Experts
8.5 Conclusion

In concluding this chapter, we confidently highlight the culmination of our empirical results for the presence of learning activities in FLOSS and the extent to which they form part of discussions and exchanges in this ecosystem. Each of the chosen repositories to analyze for tracing learning activities according to the corresponding learning phase has provided significant insights to support our hypotheses.

Mailing archives and IRC messages proved to be the right choices for the detection of process maps as they represent the learning process through its first two phases. In these phases, the role of the Novice appears to be the most active justifying the high intensity for the quest for knowledge. Equally, the Expert progressively intensifies his involvement as depicted in the trend from Initiation to Progression phases.

Regarding the last phase described in this chapter, the choice of the three datasets largely explains how learning occurs at this stage. Both the Bug reports and the Source code demonstrate the commitment of the Novice to seek answers and interact as much as possible in strengthening the acquired skills. With a participation of 49.22% for the Novice against 46.72% for the Expert and 46.19 % against 42.04% respectively on Bug reports and Source code, the Novice still engages significantly in learning. On the last dataset, Reviews, we notice an increase in the Expert’s role. The Expert performs activities to the tune of 40.36 % of total number of activities against 22.17 % for the Novice. The remaining is covered by any participant’s activities outside of the identified learning activities as labeled by “Participate in Discussions”. This significant increase in the involvement of the Expert supports our conclusion about the Expert who is more involved with reviewing the Novice’s work.

The next chapter deals with the last purpose of our global experiments by conducting conformance analysis.
CHAPTER 9: VALIDATION OF LEARNING PROCESS MODELS: CONFORMANCE CHECKING

9.1 Introduction

In the previous chapters, we presented an approach for tracing activities in learning processes in FLOSS through a number of critical steps. A number of repositories ranging from Mailing archives, IRC messages, Source code, Reviews and Bug Reports have been used to test our approach and find learning traces. We produced process maps and detailed the degree to which learning occurs in FLOSS through relevant statistical information. Moreover, in Chapter 3 we had introduced Workflow Petri nets that tentatively express how learning processes are perceived to occur in FLOSS environments. These nets constitute our a-priori models against which we test our findings. The core objective in this Chapter is to validate these Workflow Petri nets with regard to the actual behavior (processes) observed from Event Logs.

In accordance with process mining techniques, validating models requires checking their conformance with the actual exhibited behavior. In Chapter 1, we described three key outcomes of process mining as conformance, discovery and extension. Conformance checking, also referred to as conformance analysis or evaluation, constitutes the last part of this work. Rozinat and Van der Aalst [85] underline the role of conformance analysis as the technique used to check conformity between models. Given the existence of predefined models that specify how the processes should (or are expected to) be executed, conformance helps determine at what extent these models relate to the actual process models generated from recorded data (Event Logs). Paszkiewicz [128] distinguishes between a de jure process model, which specifies how things should be done or handled in order to steer and control reality and a de facto model which captures reality through Event Logs.

Based on a combination of conformance verification results, the a-priori and final models, we retrieve and contextually describe learning patterns from the FLOSS repositories considered for our experiments. This description is achieved on 2 levels. On one hand, the results of the conformance analysis provide the context in terms of occurrence of learning processes in FLOSS in light of a-priori models. Through replaying every trace in the logs on the a-priori models, conformance analysis provides an indication on how appropriately the a-priori models capture the occurrence of learning processes in FLOSS. On the other hand, the comparison between the final or a-posteriori models, generated from Event Logs through inductive mining, and a-priori models provides more insights on the similarities and differences between the two artifacts. This is summarized in Figure 224.
Therefore, in this chapter we consider the Workflow Petri nets introduced in Chapter 3, which were developed to describe the learning activities flow in FLOSS (de jure models) and perform a conformance analysis to verify how the behavior captured in the logs plays out on the de jure models. We also describe the similarities or lack thereof between the process models (de facto) as captured in the constructed Event Logs and the a-priori or de jure models.

9.2 Conformance Analysis of Learning Process models

A key aspect of conformance analysis is the ability to assess the match between two compared artifacts. This introduces and puts a clear focus on the need for metrics. Hence, depending on what the objectives of the analysis are, a number of studies have developed rules or customized metrics on their models to verify them through conformance analysis.

Some of these studies include the work by Paszkiewicz [128]. The case study used in his work concerns the verification of a number of rules on inventory processes from a Warehouse Management System (WMS) to check for conformance between discovered process and initial models. Similar experiments have been conducted with different metrics in areas such as security [84], business alignment [86], genetic mining [138] and web services [139].

Furthermore, numerous conformance algorithms and approaches have been proposed in many other areas over the years [85,123-131]. While these algorithms provide a way of checking certain properties from process models, not all of them can be useful to our analysis. Instead for our requirements, two fundamental aspects are to be verified and explained. Comparing the WF Petri nets introduced in Chapter 3 with the process maps obtained in the last few Chapters, we look at how different they are and what the most realistic way to represent learning processes in FLOSS is, in light of our conducted experiments. At the heart of such analysis is the notion of fitness. Analyzing the fitness of models is the most dominant factor in conformance [85]. Specifically, the dimensions of our endeavor at this level are twofold:

![Diagram](image-url)
— First, “Do the WF Petri nets conform to the learning processes observed in Event Logs?” The answer to this question will provide evidence and formal details regarding their level of similarity or inconsistencies thereof. Detecting discrepancies and quantifying them would help gather detailed insights into the extent to which learning processes have been theorized in comparison with reality. This quantification is attained through a set of metrics and in essence fitness. Measuring the fitness between the models and Event Logs implies verifying whether the a-priori models comply with the control flow or fit the control-flow recorded in Event Logs [85].

— Second, given the fitness, another dimension regarding the appropriateness of the models [85] is raised. “Do they (models) describe the reality or observed processes in a suitable fashion?” How appropriate are these WF Petri nets in describing learning processes given the data we have recorded? Considering the frequency of occurrence for certain parts in the processes, what can we say is the most “suitable” way to express these learning processes after such an analysis? We will consider both the structural and behavioral perspective while measuring the appropriateness of the models.

The completion of this endeavor will hopefully shed lights in establishing some form of Learning Process (LP) Alignment between WF Petri nets as expressed in Chapter 3 and the process models discovered from constructed Event Logs.

9.3 Conformance Analysis Approach

Our analysis approach implies the use of the Process Mining Toolkit [140] and the relevant plug-ins. Like numerous of other examples and studies that have utilized this toolkit [84-85,129-138], we consider in particular the Conformance Checker. Hence, the steps of our methodology will be verified and applied using this ProM plug-in.

The Conformance Checker is used for conformance analysis in order to clearly quantify our analysis metrics and locating any mismatch or discrepancy if any. The two main metrics include the fitness and the appropriateness (behavioral and structural) between the models.

— Fitness can be formally understood as the primary conformance metric that gives an indication about the extent to which the sequence traces in our Event Logs can be associated with execution paths specified by the WF Petri nets. We shall thus determine that a Petri net and Even Log fit if the net can produce all the traces in the Log [85]. This can be quantified. For example, an analysis could yield a fitness of 0.72 between a Petri net and an Event Log. A full fitness between them yields 1. However this metric on its own is not sufficient, one needs to determine how realistic any fitness is. This is achieved through appropriateness.

— Appropriateness is the second metric that specifies the extent to which the Petri net describes or not the processes behavior captured through the Event Log. It is measured at two different levels. Structural appropriateness determines the degree to which a Petri net’s structure clearly reflects the described behavior (captured flow from log traces), while Behavioral appropriateness analyzes the extent to which it represents as closely as possible what actually takes place [141] as captured by the Event Logs.
9.4 Preliminaries

On the road to perform the actual analysis experiments, we note a number of details, steps and elements we ought to emphasize and clearly define. Some of these have already been alluded to in this Chapter in a disparate fashion. However, to put everything into perspective we briefly summarize and assemble all of these prerequisite elements in this section. The first two elements are WF Petri nets and Events Logs; these are the backbone of the analysis and need to be there for any analysis to be carried out. The last element is the Mapping step.

9.4.1 A-priori Models

In Chapter 3, we produced variations of Petri nets called Workflow Petri nets that give a graphical and sequential description of learning processes in FLOSS. Figures 38 - 42 specify, based on our segmentation of the learning process in phases, learning activities pertaining to every respective phase. We highlighted the motivation with regard to the choice of this specification (WF Petri nets) for learning activities in the context of Process Mining. Most importantly, a key factor is the ability to analyze the WF nets specification with the Conformance Checker.

It is therefore in the same context that we pinpoint a few characteristics in these nets that are paramount for the analysis. Apart from their structural traits, i.e., transitions, places and tasks, the flow of tasks execution in Petri nets can be effectively interpreted by the Conformance Checker. The structural constraints and other characteristics of these WF nets are critical in analyzing processes. Van der Aalst et al. [119] argue that these nets are the class of sound WF-nets. This implies that the Petri net is required to have a single Start place and a single End place, forming paths that connect all the Nodes (Activities). Furthermore, the net is expected to define processes with dedicated Begin and End points [85,119]. Every process in the net should be able to start and complete so that all the tasks are potentially executed. We should confirm here that the models in Figures 39 - 43 in Chapter 3 meet all of these characteristics. They have single Start and End points; all tasks are executable on the paths that form the nets.

9.4.2 Event Logs

The Event Logs we make use of during this conformance analysis phase record the real behavior of interactions and executed FLOSS activities from OpenStack [163]. With five different repositories considered, we have a number of distinct logs that capture the execution of learning activities for both the Novice and Expert from every repository according to the learning phase.

Moreover, we note that all the process maps that we have considered in the previous Chapters are graphical representations of observed learning behaviors from these logs. While such process maps give an indication of the most frequent paths or processes executed, not all the paths are graphically represented less we have a spaghetti-like model as illustrated in Figure 225. However, all the paths are recorded in the Event Logs and therefore can be compared to the paths produced in the WF Petri nets in order to verify fitness.
Figure 225. Spaghetti-model representation for Novice in Progression phase on Mailing archives
Furthermore, it is critical to note that Event Logs are preprocessed in such a way that they represent traces at an aggregate level for conformance analysis. This simply implies that all process instances that have the same sequence of events (same flow of activities and additional attributes) are grouped together and combined as a logical log trace containing the number of combined traces [85]. For example if an x number of participants’ execution of activities is identical, all of these traces are grouped as one but the accumulated number of times it has been executed is also shown to indicate the value of the trace. In Figure 226, a selected trace of activities that has been followed 10665 times for an Expert on bug reporting is shown.

We have a total of 14 logs at our disposal needed for our analysis. Each log records details of every Learning participant across all three phases on different repositories. Namely we have 4 logs during the initiation phase where two repositories (Mailing archives and Internet Relay Chat Messages) are analyzed; 4 additional logs in the progression phase for the same repositories and finally 6 logs for the last phase as we use the source code, reviews and bug reports datasets to conduct our experiment.

After filtering all these logs, we can then execute the second step by pairing the activities with those produced from WF Petri nets as described in the following section.

9.4.3 Mapping

The first real step towards performing the analysis is making sure that the tasks (activities) in our Petri nets are associated with the logged events. According to Rozinat et al. [85], such pairing may result in a number of other constructs apart from the simple 1-to-1 mapping where each task in the model is uniquely associated with the corresponding activity from the log. These constructs include:

- Duplicate Tasks: This is possible when multiple tasks from the model are associated to the same type of log event because their occurrence cannot be distinguished in the log although different [85]. This can only be observed after mapping.
— Invisible Tasks: These are tasks that cannot be found in the log traces because they are not logged. This can happen in business environments, for example when certain steps are not observable, i.e., making a phone call [85]. Sometimes these are also introduced for routing purposes. However, this will not be of a great concern in our analysis as we focus on everything we define to be observable and that has been recorded.

In conformance checking, it is assumed after pairing that log events not associated with any tasks from the model are to be removed before the analysis [85]. This is done to ensure that both the model and the Event Log are at the same level of granularity [85]. However, in this step two auxiliary metrics can be defined to measure the interplay extent from both the log and model-based perspective respectively.

Metric 1 (Log Coverage) [85] Given a set of log entries $E$, a set of tasks $T$, and a set of labels $L$, let $l_E \in E \rightarrow L$, $l_T \in T \rightarrow L$, $TV = \text{dom}(l_T)$, $LT = \{ l_T(t) \mid t \in TV \}$, and $LE = \{ l_E(e) \mid e \in E \}$. The log coverage metrics, $c_E$ and $c_{LE}$ are defined as follows:

$$C_E = \frac{|\{ e \in E \mid l_E(e) \in LT \}|}{|E|} \quad (1)$$

$$C_{LE} = \frac{|LE \cap LT|}{|LE|} \quad (2)$$

Since it is possible that in the model invisible tasks are unlabeled, the mapping between tasks and labels is represented by a partial function ($\rightarrow$) [85]. Therefore, the subset of tasks that are labeled (i.e., they are in the domain of $l_T$) is precisely the set of visible tasks (i.e., $TV = \text{dom}(l_T)$). $LE$ and $LT$ denote the set of labels covered by the log and the model, respectively[85].

If we assume that $|E| > 0$ and $|LE| > 0$, then these metrics range from 0 (in the case that none of the log entries is associated to any task in the model) to 1 (when every log entry is associated to at least one task in the model) [85]. The metric $c_E$ really quantifies the overlap on the log entry level, i.e., if, for example, only one type of log entry is not covered by the model but happens very often in the log, then this metric will reflect this, whereas metric $c_{LE}$ measures the degree of overlap with respect to the types of log entries only [85].

Metric 2 (Model Coverage) Given a set of log entries $E$, a set of tasks $T$, and a set of labels $L$, let $l_E \in E \rightarrow L$, $l_T \in T \rightarrow L$, $TV = \text{dom}(l_T)$, $LT = \{ l_T(t) \mid t \in TV \}$, and $LE = \{ l_E(e) \mid e \in E \}$. The model coverage metrics, $c_T$ and $c_{LT}$ are defined as follows:

$$C_T = \frac{|\{ t \in TV \mid l_T(t) \in LE \}|}{|TV|} \quad (3)$$

$$C_{LT} = \frac{|LT \cap LE|}{|LT|} \quad (4)$$

If we assume that $|TV| > 0$ and $|LT| > 0$, then these metrics range from 0 (in the case that every visible task in the model did not occur at all in the log) to 1 (in the case that each visible task
occurred at least once in the log). Note that the metric $c_T$ quantifies the overlap on the task (or transition) level, whereas metric $c_{L_T}$ measures the degree of overlap with respect to the different types of task labels only (i.e., it is abstracted from duplicate tasks) [85].

We make use of ProM 5.2 where these metrics are implemented to carry out the mappings. We have mapped every one of the identified logs to the respective a-priori model. For illustrative purposes, we show a snapshot of the mapping between the Novice Event Log and WF Net during the initiation phase on Mailing archives in Figure 227.

![Figure 227. Mapping the Novice Log in Mailings with the corresponding WF nets](image)

We have decided to keep the labels unchanged for consistency. We refer to every activity as initially set in both the Event Log and the WF net.
9.5 Measuring Fitness and Appropriateness

In conducting our conformance analysis, the objectives are also to locate and spot differences between the predefined models and the real observed behavior recorded in logs. This can be systematically done through the use of a number of metrics depending on the purpose of the analysis in line with the number of parameters to be checked. In our context, only two principal metrics are to be measured for our analysis. We measure fitness and appropriateness to compare different model-log combinations (mappings) in order to estimate the similarity and potential deviations between the two artifacts if any [85].

Before defining these metrics, it is important to note two important factors about conformance checking. According to Rozinat et al. [85], a conformance problem can always be viewed with two assumptions. The first assumption is that a model is considered “correct” representing a process as it is expected to unfold. The second assumption supports the idea that the Event Log may be perceived “correct” since it is a reflection of real behavior. This may be justified by the fact that the model might either be outdated or not properly tailored to the performers of the tasks. Therefore, the results of the analysis may lead to the redesign of the model in order to ensure that the model and Event Log correspond to each other.

Furthermore, it is critical to highlight that the results of a conformance analysis can be the start of other types of analyses. This can trigger more improvements, refactoring or redesign of either the models or the processes recorded in the logs. Quantitative data such as probabilities, bottlenecks or even frequencies of tasks occurrence can provide more insights and indications with regards to areas to focus for a thorough analysis or areas that confirm the types of activities executed correctly etc. With respect to learning processes from FLOSS, the idea is to formulate an opinion after reenacting real behavior from data in comparison with theoretical findings from the literature. We expect that these results surely open doors for further studies in terms of applying Process Mining in analyzing FLOSS repositories for other challenges or topics that are relevant to FLOSS communities.

9.5.1 Fitness

In measuring the fitness between a model and the corresponding Event Log, every trace in the log is replayed on the model starting from the initial place. Then transitions from the logged events are fired successively according to the ordering in the trace. While replay progresses, we count the number of tokens that had to be created artificially (i.e., the transition belonging to the logged event was not enabled and therefore could not be successfully executed) and the number of tokens that were left in the model, which indicate that the process was not properly completed [85].

**Metric 3 (Fitness) [85]** Let \( k \) be the number of different traces from the aggregated log. For each log trace \( i \) (\( 1 \leq i \leq k \)), \( n_j \) is the number of process instances combined into the current trace, \( m_j \) is the number of missing tokens, \( r_j \) is the number of remaining tokens, \( c_i \) is the number of consumed tokens, and \( p_i \) is the number of produced tokens during log replay of the current trace. The token-based fitness metric \( f \) is defined as follows:

\[
    f = \frac{1}{2} \left( 1 - \frac{\sum_{i=1}^{k} n_i m_i}{\sum_{i=1}^{k} n_i c_i} \right) + \frac{1}{2} \left( 1 - \frac{\sum_{i=1}^{k} n_i r_i}{\sum_{i=1}^{k} n_i p_i} \right)
\]

It can be noted from this definition that, for all \( i \), \( m_i \leq c_i \) and \( r_i \leq p_i \), and therefore \( 0 \leq f \leq 1 \). Note also that \( c_i \) and \( p_i \) cannot be 0 because during log replay there will be always at least one token produced.
for the *Start* place and one token consumed from the *End* place [84-86]. Figures 228 and 229 are illustrations of the reply for two separate traces. In Figure 228, a total of 7 tokens are produced and are all consumed because the trace can be found on the model exactly as specified in the Event Log while in Figure 229, we show different steps of the replay and the status of the token count at every place. There is a token missing and hence one remaining because task H is not on the path between C and D.

![Figure 228](image_url)

*Figure 228. An example of Log replay for trace <A, B, D, E, A> with 0 missing tokens, 0 remaining tokens but with 7 produced tokens and 7 consumed tokens*
Figure 229. An example of Log replay for trace <A, C, H, D, F, A> with 1 missing, and 1 remaining tokens.
As an illustration in measuring this metric on our datasets, we show in Figure 230 the fitness for the Novice log (Novice_log) on the corresponding WF net (Novice_Net) in the Initiation phase as a result of our experiment. The details about all the metrics and obtained results for the 14 logs concerned in this study is given in Table 15.

Figure 230. Fitness for Novice on Mailing archives during Initiation Phase

The fitness is obtained through the function \( f(Novice\_Net, Novice\_log) = 0.045625925 \). This low value indicates that the behavior observed in the log is extremely different from the model in terms of the control-flow. Figure 230 shows a representation of the WF net used during the Event Log replay. The highlighted activities represent tasks that were missing in certain traces leading to creation of artificial tokens and recording of missing tokens. The numbers in each place indicates, after a token count, according to the (+) or (–) signs the number of remaining tokens and those of not enabled tasks accordingly. For example the number “+3118” indicates that 3118 tokens remained \((r = 3118)\) while the number “–3118” indicates the number of missing tokens.

For every event in the log trace, the transition is fired regardless of whether the current path is followed or not. Hence, the high values indicated in the model. These values also give indications of any possible mismatch. This is easily done by looking at missing and remaining tokens during log replay. We give a more detailed observation for every log in a subsequent section when we interpret all our results.

9.5.2 Appropriateness

In the event that some mismatch or discrepancies are noted on the a-priori models with the real behavior captured from the log, it is sometimes fitting to adapt the models with the actual behavior. Moreover, in the event that a few logs are a complete fit, it is also important to verify how good that can be. We can consider and determine whether a model can be more appropriate from a behavioral perspective (it does not allow for extra behavior in comparison with other models [maybe mailings archives and IRC], and from a structural perspective (it is more compact and clearly reflects the behavior observed in Event Log) [85].

Therefore, two dimensions of appropriateness can be studied with regard to conformance checking, namely behavioral and structural appropriateness [84-86]. The behavioral appropriateness also indicates the level of precision for the behavior observed in the Event Log as it is meant to be captured in the log.

9.5.2.1 Behavioral Appropriateness

In essence, behavioral appropriateness evaluates how much behavior is allowed by the model which was actually never used in the observed process executions in the log [85]. The point is to make sure that a process is modeled as precisely as possible and that the model does not allow for more behavior than necessary in order to keep it informative. This can be noted in the context where for example a model fits a log but allows for some tasks to repeat. This could trigger further executions that are not warranted and unnecessary that could impose some cost constraints.
The behavioral appropriateness can be measured according to 2 approaches. A first approach consists in determining the average number of enabled transitions during log replay [85]. The motivation is the assumption that an increase of alternatives or parallelism corresponds to a direct increase of potential behavior and thus culminating into a higher number of enabled transitions during this process. This is done by means of a metric called Simple Behavioral Appropriateness defined below.

**Metric 4 (Simple Behavioral Appropriateness)** [85] Let \( k \) be the number of different traces from the aggregated log. For each log trace \( i \) (1 ≤ \( i \) ≤ \( k \)), \( n_i \) is the number of process instances combined into the current trace, and \( x_i \) is the mean number of enabled transitions during log replay of the current trace (note that invisible tasks may enable succeeding labeled tasks but they are not counted themselves). Furthermore, \( TV \) is the set of visible tasks in the Petri net model. The simple behavioral appropriateness metric \( a_B \) is defined as follows:

\[
a_B = \sum_{i=1}^{k} n_i \frac{(|TV| - x_i)}{(|TV| - x_i) \cdot \sum_{i=1}^{k} n_i}
\]

Assuming that \(|TV| > 1\), this metric ranges from 0 (if all visible tasks in the model are always enabled during log replay. Hence, after computing for all our models and corresponding pairings, the corresponding results are given in Table 15.

Rozinat et al., [85] argue that this metric is limited in expressing behavioral appropriateness because it can only be used as a comparative means, since it measures the appropriateness relatively to the degree of model flexibility. This means that one can argue after measuring this property for two different models that one is better than the other because it allows less behavior, and the less behavior is allowed by the model the better. Moreover, it only reaches the value 1 in a purely sequential model, where exactly one task is enabled in each step of the log replay[85]. Additionally, this metric does not perform well and is unstable in cases where tasks sequences for a model include duplicate tasks.

Therefore, to resolve this constraint the potential behavior specified by the model must be analyzed and compared with the behavior actually needed to describe what was observed in the log. A set of relations can be derived from the labels that link both the tasks in the model and activities in the logs [85]. These are “Follows” and “Precedes” relations between activities from both a model and a log perspective. Looking at the tasks and activities sequences, we can determine whether two activities \((x, y)\) either always, never, or sometimes follow or precede each other as follows.

**Definition 1 (Follows relations)** [85] Two activities \((x, y)\) are in “Always Follows”, “Never Follows”, or “Sometimes Follows” relation in the case that, if \(x\) is executed at least once, then always, never, or sometimes also \(y\) is eventually executed, respectively.

**Definition 2 (Precedes relations)** [85] Two activities \((x, y)\) are in “Always Precedes”, “Never Precedes”, or “Sometimes Precedes” relation in the case that, if \(y\) is executed at least once, then always, never, or sometimes also \(x\) was executed some time before, respectively.

It is to be noted that these relations are defined as soon as they hold for any pair of labels in a sequence. As an illustration, we consider a simple sequence \((a, \ldots , a, \ldots , c, \ldots , a)\). In this case, the pair \((a, c)\) is an element of the “Follows” relation, although it does not hold for all \(a\) that they are eventually followed by \(c\) [85].
Furthermore, it is possible to express “Sometimes” relations in order to capture variability in behavior. An illustrative case would be a situation where activities preceding a number of alternative branches are sometimes followed by one of these alternative branches and sometimes by another. The same occurs for activities that follow after a number of alternative branches were joined. Therefore, the idea of the following metric is to compare the variability of the behavior allowed by the model and the behavior observed in the log based on the cardinal numbers of the $SF$ and $Sp$ relations [85].

**Metric 5 (Advanced Behavioral Appropriateness) [85]** Let $S^m_F$ be the $SF$ relation and $S^m_P$ be the $Sp$ relation for the process model, and $S^l_F$ the $SF$ relation and $S^l_P$ the $Sp$ relation for the Event Log. The advanced behavioral appropriateness metric $a'_B$ is defined as follows:

$$a'_B = \left( \frac{|S^l_F \cap S^m_F|}{2 \cdot |S^m_F|} + \frac{|S^l_P \cap S^m_P|}{2 \cdot |S^m_P|} \right)$$

It can be noted that this metric helps capture situations where—according to the model—two activities may sometimes follow each other (and sometimes not), but in the log they always or never follow each other. The reverse can also happen, i.e., the model is more specific than the log, which then indicates a fitness problem. Also, although the $SF$ and $Sp$ relations are symmetric, we consider both and weigh them equally to make the metric stable with respect to the position of the “extra behavior”.

![Figure 231. Behavioral appropriateness or precision for Novice Model in Initiation on Mailing](image)
9.5.2.2 Structural Appropriateness

The basic idea about structural appropriateness in the context of this analysis pertains to the control flow perspective, and often there are several syntactic ways to express the same behavior in a process model [85].

As we explore these metrics, it is important to highlight here that modeling a business process in a meaningful way is difficult to capture by measurement [85]. Therefore, determining the suitability of any models can be regarded as a matter of preferences in light of the objectives of the analysis. Other aspects such as the details of described workflow actions may play a role in the way the models are interpreted. Some models might exhibit conformance problems in their structure with the existence of unwarranted constructs such as duplicate tasks, invisible tasks, and implicit places. These can impact the execution of processes and affect the overall performance of the model.

Hence, the detection of such potentially problematic constructs is paramount in understanding and analyzing the process models within the context of our objectives. This can be accomplished through a number of metrics. First, a simple metric based on the number of different task labels in relation to the graph size of the model is defined.

**Metric 6 (Simple Structural Appropriateness) [85]** Let \( L \) be the set of labels that establish the mapping between tasks in the model and events in the log, and \( N \) the set of nodes (i.e., places and transitions) in the Petri net model. The simple structural appropriateness metric \( a_S \) is defined as follows:

\[
a_S = \frac{|L| + 2}{|N|}
\]

Given the fact that a WF-net is expected to have a dedicated Start and End place, the graph must contain at least one transition for every task label, plus two places (the start and end place). In this case \(|N| = |L| + 2\) and the metric \( a_S \) yields the value 1. The more the size of the graph is growing, e.g., due to additional places, the measured value moves towards 0.

The value indicates how the two artifacts match in terms of internal structure etc. For example, a value of \( a_S(M_{Net}, M_{Log}) \approx 0.170 \) is a very bad value caused by the many duplicate tasks (as they increase the number of transitions while having identical labels).

However, this metric can only be used as a comparative means for process models that exhibit equivalent behavior (because it is only based on the graph size of the model). Therefore, it is of limited applicability. Thus, a better way to do this is through following a new structural appropriateness approach. As a design guideline, constructs such as alternative duplicate tasks (duplicate tasks that never happen together in one execution sequence) and redundant invisible tasks (invisible tasks that can be removed from the model without changing the behavior) should be avoided as they were identified to inflate the structure of a process model and to detract from clarity in which the expressed behavior is reflected.
A more complete description including a formal specification of the approach can be found in a technical report [142]. Note that because the number of paths in the model can become very large, the cost of detecting alternative duplicate tasks may be problematic. In contrast, redundant invisible tasks can be detected via structural analysis of the model, which is typically very fast.

**Metric 7 (Advanced Structural Appropriateness)** Let $T$ be the set of transitions in the Petri net model, $T_{DA}$ the set of alternative duplicate tasks, $T_{IR}$ the set of redundant invisible tasks. The advanced structural appropriateness metric $a'_S$ is defined as follows:

$$a'_S = \frac{|T| - (|T_{DA}| + |T_{IR}|)}{|T|}$$

Note that $|T_{DA}| + |T_{IR}| \leq |T|$ as duplicate tasks are always visible and therefore $0 \leq a'_S \leq 1$.

![Figure 232. Structural appropriateness or precision for Novice Model in Initiation on Mailing](image)

In Figure 232, the values for both structural properties are given. In terms of structure, $a_S(\text{Novice\_Net}, \text{Novice\_log}) = 0.5294118$ and $a'_S(\text{Novice\_Net}, \text{Novice\_log}) = 0.6666667$. These values indicate at what extent the model pair with the structure of the log for this process. This is useful to identify where duplicate values are and this could lead to any relevant decision depending on how the structure influences the role of the model etc.

With regard to Learning Processes in FLOSS, it is not critical to be strict on the structure as the environment is expected to be dynamic and unpredictable but these values give an indication of how one can understand the structure of a learning process graph in the instance that FLOSS members engage into learning activities. In section 9.6, we give all the values for our logs and interpret them in light of the objectives outlined in this thesis before we conclude in Chapter 9.
9.6 Interpretation of Conformance Results

According to Rozinat et al.[85], these conformance measures are orthogonal to each other and provide different insights as they measure something completely different from each other. This implies that a level of good behavior or structure does not necessarily imply a good fitness and vice versa.

However an analysis based on all of these measures surely provides some indications with respect to the overall performance of the model as compared to the Event Log. More importantly, these metrics help us explain how the occurred real learning behavior differs from the initial description that is provided so far in the literature and presented in Chapter 3. Table 15 summarizes all the values according to the 14 Event Logs we have across the different repositories in the three learning phases for all the participants.

9.6.1 Primary Observations

The primary observation to recall here is that, the descriptive process models of learning behavior in FLOSS environments need to be understood from a totally different perspective than it would be for normal business processes in a business environments. As a dynamic setting where people’s movements and involvement cannot be totally predicted and controlled, understanding the processes that take place in FLOSS environments requires that we look at them in the appropriate context. Many users are actively engaged in numerous activities ranging from simple social interactions to more engaged knowledge exchanges. If the data at our disposal gives a glimpse of such happening, we use such results to explain at what extent learning is expected to happen in a general factual sense.

Therefore two main general observations can be drawn from the conformance analysis conducted on our a-priori models with regard to the corresponding logs. Looking at the results in Table 15, we can globally observe that:

1. These results provide a comfortable level of confidence for our FLOSS mining approach. The steps we undertook to process the repositories, apply process mining and produce process models have produced almost the same results in terms of model behavior for a number of logs in different phases and repositories. In particular, the advanced behavioral and structural appropriateness for the Novice_Log in both Mailing archives and Internet Relay chats in the initiation phase are exactly the same with values respectively \( a_B = 0.0, a_S = 0.5294118 \) and \( a'_S = 0.6666667 \) while this is also true for the simple and advanced structural appropriateness with regard to the Expert_Log in the same phase and on the same repositories with respective values of \( a_S = 0.72727275 \) and \( a'_S = 1.0 \).

The same pattern can be noticed in the second phase (progression) where the Novice_Log in both repositories yields respectively \( a_S = 0.5555556 \) and \( a'_S = \).
Table 15. Conformance Analysis Results in FLOSS Repositories

<table>
<thead>
<tr>
<th>Phase</th>
<th>Repository</th>
<th>Novice Log</th>
<th>Expert Log</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( f = 0.045625925 )</td>
<td>( f = 0.58433735 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B = 0.8492753 )</td>
<td>( a_B = 0.63818514 )</td>
</tr>
<tr>
<td></td>
<td>Mailing archives</td>
<td>( a_B' = 0 )</td>
<td>( a_B' = 1.0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S = 0.5294118 )</td>
<td>( a_S = 0.72727275 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S' = 0.6666667 )</td>
<td>( a_S' = 1.0 )</td>
</tr>
<tr>
<td>Initiation</td>
<td></td>
<td>( f = 0.3585199 )</td>
<td>( f = 0.5337318 )</td>
</tr>
<tr>
<td></td>
<td>Internet relay Chats</td>
<td>( a_B = 0.6624337 )</td>
<td>( a_B = 0.87003195 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B' = 0 )</td>
<td>( a_B' = 0.8333333 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S = 0.5294118 )</td>
<td>( a_S = 0.72727275 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S' = 0.6666667 )</td>
<td>( a_S' = 1.0 )</td>
</tr>
<tr>
<td>Progress</td>
<td>Mailing archives</td>
<td>( f = 0.15142721 )</td>
<td>( f = 0.26572186 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B = 0.64597225 )</td>
<td>( a_B = 0.9053976 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B' = 0.39285713 )</td>
<td>( a_B' = 0.93826085 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S = 0.5555556 )</td>
<td>( a_S = 0.5 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S' = 0.7777778 )</td>
<td>( a_S' = 0.5 )</td>
</tr>
<tr>
<td></td>
<td>Internet relay Chats</td>
<td>( f = 0.10818714 )</td>
<td>( f = 0.1161989 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B = 0.69343656 )</td>
<td>( a_B = 0.663768 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B' = 0 )</td>
<td>( a_B' = 0.37608695 )</td>
</tr>
<tr>
<td></td>
<td>Bug Reports</td>
<td>( a_S = 0.5555556 )</td>
<td>( a_S = 0.5 )</td>
</tr>
<tr>
<td></td>
<td>Reviews</td>
<td>( a_S' = 0.7777778 )</td>
<td>( a_S' = 0.5 )</td>
</tr>
<tr>
<td>Maturation</td>
<td>Source Code</td>
<td>( f = 0.52500534 )</td>
<td>( f = 0.5210679 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B = 0.77798 )</td>
<td>( a_B = 0.7829485 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B' = 1.0 )</td>
<td>( a_B' = 1.0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S = 0.7037037 )</td>
<td>( a_S = 0.6060606 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S' = 0.8888889 )</td>
<td>( a_S' = 0.8947368 )</td>
</tr>
<tr>
<td></td>
<td>Source Code</td>
<td>( f = 0.43482816 )</td>
<td>( f = 0.40341926 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B = 0.827901 )</td>
<td>( a_B = 0.76634383 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B' = 0.99593496 )</td>
<td>( a_B' = 0.992 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S = 0.7037037 )</td>
<td>( a_S = 0.6060606 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S' = 0.8888889 )</td>
<td>( a_S' = 0.8947368 )</td>
</tr>
<tr>
<td></td>
<td>Source Code</td>
<td>( f = 0.4751042 )</td>
<td>( f = 0.51980984 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B = 0.6931343 )</td>
<td>( a_B = 0.6099818 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_B' = 0.63986015 )</td>
<td>( a_B' = 0.13308084 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S = 0.7407407 )</td>
<td>( a_S = 0.6060606 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a_S' = 1.0 )</td>
<td>( a_S' = 0.8947368 )</td>
</tr>
</tbody>
</table>
0.7777778 while the Expert_Log in the same repositories produce $\alpha_S = 0.5$ and $\alpha_S' = 0.5$ when it comes to the structural appropriateness of the a-priori model in light of observed behavior. In the last phase, we can also point that the same pattern observed on both the bug reports repository and reviews while these values are slightly different from the source code due to the additional activity that can only be found on this repository and not in the other two (Submit Documentation (N-C)) by the Novice. This is because as the novice commits some code, there is a detailing document that accompanies such submissions which is an exclusive task that can only be accomplished with source code and probably not with both reviews and bug reporting. These values are respectively for the Novice_Log for bug reports and reviews $\alpha_S = 0.7037037$, $\alpha_S' = 0.8888889$, while for source code one can observe slight differences with $\alpha_S = 0.7407407$ and $\alpha_S' = 1.0$ for both the simple and advanced structural appropriateness. However for the Expert_Log, we notice that in terms of structural appropriateness all the three logs provide the same results specifically $\alpha_S = 0.6060606$ for the simple structural appropriateness and $\alpha_S' = 0.8947368$ for the advanced metric.

2. Secondly, even in terms of fitness all the logs do not really fare far from each other’s results with minor differences. The slight variations in terms of fitness could be explained by the size of each repository in light of the number of traces that were analyzed. Since conformance analysis is conducted in terms of log replay, the fitness of a model on a log is greatly dependent on the number of tokens produced, as they are consumed, remaining or missing during this replay. So, the larger the repository, the larger the possibility of having some variations in terms of fitness. Nevertheless, we can note that the fitness for all the logs shows that there is a discrepancy between how learning processes are projected to occur in FLOSS and the actual way in which they do as expressed in the log. This discrepancy is viewed from the control-flow perspective.

In addition to these two main observations, it is helpful that we describe and contextualize although succinctly these results for each phase with regards to all the repositories and the main common behavior to retain in each phase as exhibited in the log.

### 9.6.2 A-priori Models diagnostics

#### 9.6.2.1 Initiation Phase

In the first phase of the learning process, the results of our analysis for the two repositories considered are indicative of the disparity with regards to the flow of execution as we tentatively expressed it with the WF Petri nets in Chapter 3 and the actual behavior captured in the logs for both the Mailing archives and Internet Relay chat messages for the Novice. This is observed through the value for fitness. The results indicate that the fitness for the Novice_Log on Mailing archives is $f (Novice_Net, Novice_Log) = 0.045625925$ which is very far from an ultimate 1.0 complete or full fitness if the two artifacts match. While on Internet Relay chats message, the fitness is given by $f (Novice_Net, Novice_Log) = 0.3585199$ although a little higher but way below 0.5 that could suggest a potential average match or fit between the two artifacts.

In reality, we point out that many of the traces recorded from the two logs do not potentially have the same starting point as we will observe in the Petri nets produced from the logs. While the WF net suggests that starting a learning process would logically stem from formulating a question to identifying
an Expert and so forth, the real behavior indicates that due to the dynamic nature of the FLOSS environments this cannot be entirely predicted. Sometimes people would start with a different activity and even stop in the middle depending on their interest and initial set of skills. As we considered unique traces to ensure that we have a somewhat global view of the learning process in these repositories at this phase, it is to be pointed out that as larger repositories are considered, it is possible to obtain different values for the fitness if different instances of log traces are encountered.

Furthermore, looking at the second important metric in this analysis, we consider the appropriateness of the models as they perform in light of the log traces. The first perspective concerns behavioral appropriateness and we observe that the model on both repositories for the $a_B(Novice_{Net}, Novice_{log}) = 0.0$ clearly indicates that there is a huge mismatch between the two artifacts. The value 0 indicates that the model does not at all allow for the behavior observed in the log. This is because there are certain activities of which the order of occurrence greatly deviates from reality. Figure 233 below represents the diagnostics for the Novice on both the Mailing and Internet Relay chat messages.

![Diagnostics Model appropriateness on both Mailing archives and IRC messages for Novice in FLOSS](image)

While all these activities can be observed on the logs, in some instances certain activities always follow each other i.e. Formulate Question “always follows” Post Message, Post Question and Comment Post while in some other instances Identify Expert “always follows” the same three activities. Naturally and from a theoretical point of view, one would imply that before we post a question, we must have formulated it and hence Post Question would always follow Formulate Question. However, from a behavioral point of view as observed in the log, Formulate Question always follows Post Message or Question. This is because a Post on its own does not only include a question, some other introductory
Chapter 9: Validation of Learning Process Models: Conformance Checking

...comment amounts to differentiating and characterizing the exclusive Post activity; which also indicates that the poster identifies an expert as Identify Expert always follows any of the three activities. However, for the last two activities, it interestingly seems to work perfectly as after contacting, one sends detailed request and this sequence can be correctly observed in both artifacts (WF net and Log).

The last perspective regarding the structural appropriateness helps to define what is considered to be the preferred way to express specific behavioral patterns [85] and somehow spot any discrepancies with regard to violations of these rules. With this measurement, duplicate tasks and invisible tasks are key role players and impact on the outcome. In the context of our work, we notice that the only duplicate tasks are linked to the elimination of the XOR-split/join semantics expressed in the WF nets in Chapter 3. As highlighted in Figure 234, duplicate tasks (Identify Expert) are generated to eliminate these constructs and clearly show every path during analysis for conformance purposes. While there is initially an XOR-split after this activity, this is replaced by repeating the same activity as it would take any of the options between Post Message, Post Question or Comment Post.

![Figure 234. reconstructed Petri net with alternative duplicate tasks for Novice on Mailing and IRC during initiation](image)

The final results indicate that the WF net in this instance only matches the real structural appropriateness observed in both logs with a values of $a_S (Novice_Net, Novice_log) = 0.5294118$ and $a_S' (Novice_Net, Novice_log) = 0.6666667$ respectively for the simple and advanced structural appropriateness. These are indicative of the disparity between what is projected to happen and the real behavior as observed. Nevertheless, our view is that there could not be a complete fit for this metric as it is unnecessary or impossible to predict how every single participant in FLOSS would engage in the learning process. Although, these values provide some orientations regarding how one could estimate the link between the two artifacts in light of this metric.

Figures 235-241 are representative of the behavior observed in the logs for each repository as Petri nets. Although, in the previous Chapters we gave process maps about this behavior, one should retain that those provide a general picture in terms of most frequent paths, most frequent activities etc. as recorded in the logs. This is critical because it backs at the core the importance of this study in terms of the proportions of learning activities one can observe in FLOSS. However the idea in Figures 235-241 is to depict a Petri net representation that captures the Novice’s learning behavior in this phase in FLOSS. We make use of the inductive miner plug in in ProM to generate a clear and comprehensible Petri net that is representative of the observed behavior. A key motivation in adopting this visualization is the ability to depict where all possible iterations are expected to occur and the proportionality with regard to the general performance of the model for every zone or group of activities occurring together. The choice of this tool to display the visualization of the discovered model is also justified by its ability to depict a sound model (free of deadlocks and anomalies) and its capacity of handling infrequent events [143-144]. More precisely, the Inductive Visual Miner, as described in [142] conducts a process exploration that leads to a satisfactory model.
Chapter 9: Validation of Learning Process Models: Conformance Checking

Figure 235. Final Learning Model for Novice during Initiation phase on Mailing archives with a replay of 1000 traces

Figure 236. First Group of Activities depicting first point of Iteration
Chapter 9: Validation of Learning Process Models: Conformance Checking

Process Models for Learning patterns in FLOSS repositories
Chapter 9: Validation of Learning Process Models: Conformance Checking

Process Models for Learning patterns in FLOSS repositories  Page 263

Figure 239. Final Learning Model for Novice during Initiation phase on IRC with a replay of 1000 traces

Figure 240. First Group of Activities depicting first point of iteration

Figure 241. Second Group of Activities depicting second point of iteration
We also include a start and end activities as endpoints to provide a clear generic interpretation of the model. This is achieved through a replay of 1000 traces that can give a feel of the model execution. We replicate the same framework for the rest of the models discussed and introduced in this Chapter.

The Petri net in Figure 235 depicts the Novice behavior on Mailing archives. The main characteristics that differentiate from the a-priori model are the presence of iterative points, the grouping of activities that are likely to occur together and their flow of occurrence as shown in Figures 236-238. We note from the starting point that a learning process in this phase can start from 3 different activities and follow a particular path depending on the Novice’s needs, interests and levels of experience in the community for the Mailing archives. The Model shows that on Mailing archives, a Novice can start a process from three main points. Firstly, a Novice can ContactExpert and SendDetailedRequest, showing the first block of activities that are more likely to occur together and even be repeated a number of times. The next choice includes Formulate and PostQuestion as the second block of activities occurring together; while the last choice is to IdentifyExpert, PostMessage or CommentPost a number of times as needed and complete this phase.

The same trend can be noticed in Figure 239 about the Novice behavior on Internet Relay Chat messages with the only difference lying on the starting point. A novice can start with ContactExpert and SendDetailedRequest, which can be repeated a number of times as needed and then either Formulate and PostQuestion as the second block of activities that can be iterated together before completing the phase or the last choice is to CommentPost, PostMessage and IdentifyExpert and either complete the phase or iterate through the same path a number of times or moving to the second choice before the phase is finally complete.

The last remark about the discovered models in both repositories is that different Novices can learn or engage in the learning process on different scales. Not everybody starts from the beginning and follows all the paths and activities from beginning to end. Depending on the learning objectives, the question in play and the Novice’s experience, he/she can execute a number of certain activities and not necessarily all of them. Nevertheless, these representations give a more realistic idea of how the flow of execution is likely to happen when a Novice engages in a learning process.

Looking at the Expert’s data, we notice however a totally different trend. Unlike the major discrepancies observed with the Novice in both the Mailing archives and Internet Relay Chat messages, the results on the same repositories indicate a more positive fit between the Event Logs and the WF Petri nets. The fitness for the Expert_Log on Mailing archives is \( f(\text{Expert\_Net, Expert\_Log}) = 0.5843735 \); although less than the ultimate 1.0, it is above the 0.5 average while on Internet Relay chats message, the fitness is given \( f(\text{Expert\_Net, Expert\_Log}) = 0.5337318 \) which is also above the average 0.5 suggesting a potential average match or fit between the two artifacts. Since this is a log replay perspective, we can confidently say that the WF Petri nets in this instance can be entirely verified on the Event Logs with no mismatch in terms of the control flow as demonstrated in Figure 242. A full fitness could not be attained simply because of the number of instances and size of repositories. It is possible to encounter short traces that do not cover conveniently the model paths and the fitness for each of such traces has a bearing effect on the overall fitness.
The results of the remaining metrics corroborate this observation in the sense that by taking into account both the structural and behavioral appropriateness, the Expert Log in both repositories perform significantly well. Specifically, we observe that the behavioral appropriateness for the model on Mailing archives is
\[ B(Expert_{Net}, Expert_{log}) = 1.0 \]
indicating a perfect match between the two artifacts, while on messages in Internet Relay Chats, it is
\[ B(Expert_{Net}, Expert_{log}) = 0.8333333 \]
clearly indicating almost a perfect match too but an acceptable degree of behavioral appropriateness between the two artifacts. This can be explained by the existence of certain tasks being redundant in some instances for a much reduced number of traces.

Finally, the results for the structural appropriateness observed in both repositories are given by values of
\[ S(Expert_{Net}, Expert_{log}) = 0.72727275 \] and
\[ S(Expert_{Net}, Expert_{log}) = 1.0 \]
respectively for the simple and advanced structural appropriateness. As we consider only the advanced structural appropriateness, one can clearly observe a full match between our WF nets and recorded observed behavior from Event Logs.

Looking at these results for the Expert, one can note that the results of the conformance analysis corroborate largely the WF nets with regards to the control flow in spite of the existence of short traces that hinder on the entire model coverage affecting the final full fitness. This is somewhat expected as in the previous case with the Novice Behavior because although WF models give a standardized possible description of learning processes, the real behavior is much more richer and captures different occurrences in a more dynamic fashion.

Nevertheless, the behavioral and structural appropriateness values indicate that even at its raw level, the WF net for the Expert’s contribution in the learning process during this phase appears to be a good representation. With perfect match for both values, one can expect Experts of users who respond to questions and engage in knowledge exchange interactions in FLOSS to follow the pattern captured in our WF Petri net. That is, in some instances they would start from Reading a Post and hence appropriately react to it if they can, in other instances they can start from Reading a Message from a thread or Read comment from a source code and then contact the Novice or the person seeking help before asking for further details or clarification through another Comment etc.

Taking into account these results and in light of the slight differences explained above, we give a final representation of the learning process for the Expert in this phase through the Petri nets on both repositories. Although not completely claiming to be the ultimate description of learning behavior at this level, we can at least have some confidence in assuring that it does explain entirely what occurs at a certain degree.

Figure 242. Diagnostics Model appropriateness on both Mailing archives and IRC messages for Expert in FLOSS
Figure 243. Final Learning Model for Expert during Initiation phase on Mailing archives with simulated behavior using 1000 traces
Chapter 9: Validation of Learning Process Models: Conformance Checking

Figure 244. Final Learning Model for Expert during Initiation phase on IRC with a replay of 1000 traces

Figure 245. First Group of Activities depicting first point of iteration

Figure 246. Second Group of Activities depicting second point of iteration
In Figure 243, the model depicts a representation of three possible paths that an Expert travels while taking part in the learning process from Mailing archives. Similarly to the Novice Model, the points of iteration are indicated and in this case, all three paths can be iterated accordingly. The first path indicates that the Expert starts engaging through ReadMessage, ReadPost and CommentPost. The second path is ReadSourceCode and lastly, ContactNovice and CommentPost/SendFeedback. Any of these paths if taken first can be either iterated completely and then the phase is complete or it can be directly followed by any of the other two paths or both before completing the phase.

The same behavior is observed on the Internet Relay chat messages in Figure 244 as the Expert would ReadMessage, ReadPost and CommentPost, which could be iterated a number of times as needed and then either ContactNovice and CommentPost/SendFeedback or ReadSourceCode and iterate accordingly as indicated by the two points of iteration in Figures 245 and 246 before completing the phase. As noted with the Novice, the general trend and feeling about these final models are that we cannot fully determine whether all users/ Experts would begin a learning process from the starting point and follow all the paths and activities from beginning to end. Depending on the learning objectives, the question in play and the Expert’s experience, he/she can execute a number of certain activities deemed fit to this level of expertise at any point during the process. Nevertheless, these representations give a more realistic idea of how the flow of execution is likely to happen when an Expert takes part in a learning process.

### 9.6.2.2 Progression Phase

In this second phase, we performed our analysis on the same repositories as motivated in Chapter 7. The primary observations with regards to the values obtained show practically the same trend for the Novice’s behavior described in the first phase and even for the Expert. The other critical observation is probably the fact that the same values are obtained on different logs which, once again, provides some level of confidence in the approach used while analyzing these data sets. We describe the obtained diagnostics in the following paragraphs starting from the Novice to the Expert in both repositories (Mailing archives and Internet Relay Chat messages).

After running the experiments on Mailing archives, for the first metric (fitness), the results indicate that the fitness for the Novice_Log on Mailing archives is \( f(\text{Novice Net}, \text{Novice Log}) = 0.15142721 \) which is very far from an ultimate 1.0 complete or full fitness if the two artifacts match. While on Internet Relay chats message, the fitness is given by \( f(\text{Novice Net}, \text{Novice Log}) = 0.10818714 \) suggesting that the control flow between the two artifacts occurs quite differently. This disparity in terms of log replay on the WF Petri net can be observed through the diagnostics model in Figures 247 and 248. Figure 247 indicates for a number of activities including Providing Feedback and Posting Questions during the review state, Sending Feedback, Posting Questions and Replying to Posted Questions during the post state as well as Analyzing Source code during the analysis in the analyze state always come after Reporting Bugs of the analyze state. Additionally, the same applies for Analyzing Source Code in the analyze state which is always executed before Commenting on Source Code in the same state. In turn, this task also always occurs before Sending Feedback in the post state, and Posting Questions in the review state.

This diagnostic makes complete sense as it would be suitable to be sending feedback when the interaction revolves around a bug to have reported it first before learning participants could post questions, reply to questions or even send feedback in this context. Also, before commenting on code, one would have to analyze it first. The same applies for Commenting on Code before sending a relevant feedback on code or even post questions.
Figure 247. Diagnostics Model appropriateness on Mailing archives for Novice in FLOSS during Progression Phase
Figure 248. Diagnostics Model appropriateness on Internet Relay Chat messages for Novice in FLOSS during Progression Phase
While performing the same experiments for the same participant on Internet Relay chats, one can observe that almost the same trend is observed as indicated by the diagnostic model in Figure 248. While everything observed on Mailings is applicable to this repository as well, we note some additional constraints with regard to activity Posting Questions and Providing Feedback in the review state as well as Replying to Posted Questions and Posting Questions in the post state. We note that in this data set, when these activities are executed in conjunction with the activity Comment on Code in the analysis state, all of these activities always execute before a comment is made on code especially when the thread has to do with source code except for Posting Questions in the post state which never follows the trend like the other activities.

Lastly for the Novice in this phase, we look at the appropriateness. The behavioral and structural appropriateness on Mailings are respectively given as $a_B(Novice_Net, Novice_log) = 0.39285713$ and $a_S(Novice_Net, Novice_log) = 0.7777778$. While the structure of the model is above the average 0.5, the behavioral appropriateness is quite low and as expected from the fitness results, the WF Petri net is not appropriate enough to describe how the Novice learn at this phase. Looking at the same metrics on Internet Relay Chat messages, the same trend can be observed with the behavioral and structural appropriateness respectively given as $a_B(Novice_Net, Novice_log) = 0.0$ and $a_S(Novice_Net, Novice_log) = a_S = 0.7777778$ practically the same value obtained on Mailing archives.

It is critical to note here that during the process of discovering learning activities and their order of occurrence, certain activities cannot be predicted to happen with absolute certainty. While the WF nets provided a tentative description of such behavior, we can observe much richer and precise descriptions from the real behavior. On the basis of this information, we produce the final Petri net that can be observed on any repositories where messages are exchanged as seen in Figures 251 and 252.

In Figure 249, we notice the progress made during the learning process triggered since the first phase. We note that the Novice will PostQuestions on any number of topics or discussions forums before choosing to either AnalyseSourceCode in relation to posted questions, SendFeedback when needed or simply CommentOnCode. The last step of the path involves ReportBugs or ReplyPostedQuestions, ProvideFeedback or PostQuestions when the need arises before completing the phase. All of these activities can be iterated as many times as required before moving forward. Furthermore, it can be noted that on Mailing archives, the Novice is expected to perform any of these activities at any point in time as part of this progression phase.

Considering Figure 252, the behavior exhibited by the Novice on Internet Relay Chat messages differs slightly from Mailing archives in the number of starting and finishing activities. The Novice has the choice to either PostQuestions that might have arisen as part of reviewing counterparts or Expert’s feedback, or directly SendFeedback before he/she can then PostQuestions. The last part of the path is diversified by the rest of activities. Any of the 5 remaining activities can be executed a number of times or even skipped before the phase draws to an end as shown in the groupings in Figures 253 and 254.

These Petri nets provide a graphical standard representation of how the behavior of Novice is captured in the two taken repositories based on the records we analyzed and any variation is possible if a different repository is considered. Nevertheless, in spite of any possible variations, these models largely and sufficiently describe how learning occurs in these repositories in a broad sense.
Chapter 9: Validation of Learning Process Models: Conformance Checking

Figure 249. Final Learning Model for Novice during Progression phase on Mailing archives with a replay of 1000 traces

Figure 250. First Group of Activities depicting first point of iteration

Figure 251. Second Group of Activities depicting second point of iteration
Figure 252. Final Learning Model for Novice during Progression phase on IRC with a replay of 1000 traces

Figure 253. First Group of Activities depicting first point of iteration
Looking at the Expert’s data, the results of the experiments give indications of similar behavior as Novice. It seems that during this phase, the process flow captured in the a-priori models providing a standardized view is much simpler with regards to how the learning activities are executed in reality. Hence, the fitness for the Expert_Log on Mailing archives in this phase is $f(\text{Expert Net}, \text{Expert Log}) = 0.26572186$; while on Internet Relay chats message, the fitness is $f(\text{Expert Net}, \text{Expert Log}) = 0.1161989$ suggesting a very low fitness between the two artifacts.

Since this is a log replay perspective, we can certainly say that the ordering of activities tend to differ significantly as represented on Event Logs. We can argue that a full fitness could not be attained given the variations and dynamism in the number of traces and the length of every analyzed trace. Figures 255 and 256 show diagnosis models for both repositories.
We note from Figure 255 that while executing the relevant activities during this phase, there are certain activities the Expert would execute in a particular strict order when taken together in a trace. Namely, two trends of constraints are observed including “always follows” and “never follows”. The activities Analyze Source Code and Run Source Code in the Analyze phase never follow the activity Comment on Code in the same state. This is clearly understandable in the sense both activities have got to be executed first respectively when used in conjunction with Comment on Code before the latter is executed. Finally, the same activities always execute after the activity Report Bugs in the same state. As it was the case with Novices, when the thread is about a bug, the source code needs to be analyzed or run after reporting a bug.

On the other hand, Figure 256 provides a wider description and activities flow for the Novice in Internet Relay Chat messages. We note that once again this discrepancy in fitness is due to the fact that a number of activities are set to follow precedence constraints as they occur in the same traces with other activities. Specifically, we observe that all the activities always follow the activity Report Bugs in the analyze phase every time any of them is executed in conjunction with the latter. In addition, activities Review Thread Code and Review Thread Posts in the review state also always follow activities Reply Posted Questions and Post Questions in the post state as shown with red arcs in the figure. Finally, the activity Send Reply during review state always follows Reply Posted Questions in the post state.
Figure 256. Diagnostics Model appropriateness on Internet relay Chat messages for Expert in FLOSS during Progression phase
As it pertains to precision or behavioral appropriateness, the results indicate that on Mailing archives, one can accept such a description as given through the WF Petri net with value $a'_{ID}(\text{Expert\_Net}, \text{Expert\_log}) = 0.93826085$ indicating a good match between the two artifacts, while on messages in Internet Relay Chats $a'_{IN}(\text{Expert\_Net}, \text{Expert\_log}) = 0.37608695$ clearly indicating that the volume of messages on this repository affects the overall match. The details we have provided above explain why for chat messages, the gap is big. Hence, the final Petri net we produce from the Log reflecting all these constraints identifies and provides a more descriptive learning process for the Expert during this phase.

Finally, the results for the structural appropriateness observed in both repositories are given by values of $a_{S}(\text{Expert\_Net}, \text{Expert\_log}) = 0.5$ and $a'_{S}(\text{Expert\_Net}, \text{Expert\_log}) = 0.5$ respectively for the simple and advanced structural appropriateness. One can clearly observe an average match between our WF nets and recorded observed behavior from Event Logs.

Looking at these results for the Expert, we note that the structure of the WF nets provides a 50% confidence level in terms of the structure of models describing the actions of Experts during this phase. While on Mailing archives, the behavioral appropriateness is quite high, the same does not apply on Internet Relay Chat messages as explained above. This is partly due to the size and volume of data in the latter as well as the overall number of traces considered for analysis. As expected, the fitness also indicates the existence of traces that hinder on the entire model coverage affecting the final full fitness. From these observations, we consider all these elements and results and produce global Petri nets based on the recorded data expressing the Expert’s behavior in this phase as seen in Figures 257 and 260.

In Figure 257, on Mailing archives, the Petri net obtained indicates that the Expert can follow 4 possible full paths while undertaking activities as related to this phase. First, the Expert would SendReply and then CommentOnCode, AnalyzeSourcecode and RunSourceCode before either ReportBugs or execute any of the remaining activities before existing. The second choice is to directly start with the path CommentOnCode, AnalyzeSourcecode and RunSourceCode skipping SendReply and then proceed like in the first instance. The third possibility is to skip both the first and second alternatives and just ReportBugs especially on Novice’s work before executing any of the last block of activities. And finally, the last choice depicts skipping all the first three choices and just distinctly performing the last block of activities before completing the phase. The grouping of activities can be seen in Figures 258 and 259.

Figure 260 on the other hand, shows a Petri net representing the behavior of the Expert on Internet Relay chat messages. While the behavior appears practically the same, with the first three alternatives from the Petri net on Mailing archives all traceable here, the only variation is that ReportBugs can be executed alternatively with PostQuestions right after the second block of activities and before the last block of activities prior to completing this phase.
Figure 257. Final Learning Model for Expert during Progression phase on Mailing with a replay of 1000 traces
Figure 258. First Group of activities depicting first point of iterations

Figure 259. Second Group of activities depicting second point of iterations
Figure 260. Final Learning Model for Expert during Progression phase on IRC with a replay of 1000 traces

Figure 261. First Group of activities depicting first point of iterations
Figure 262. Second and Third Group of activities depicting second point of iterations
9.6.2.3 Maturation Phase

In this last phase, the learning process has progressively grown into more technical activities over source code, bug reporting and even general applications or artifact reviews. Three repositories were used as explained in Chapter 8 to model FLOSS community members’ learning behavior. We have, thus, considered the results of the discovery process for each of these repositories for conformance analysis.

Looking at the conformance analysis results, perhaps one of the critical observations is the continuation of the trend in obtaining the same values for certain metrics on these different repositories which amplifies somewhat the effectiveness of our approach at some extent. Additionally, another critical observation is that the results also show that the WF Petri nets for both the Novice and Expert are not really far from representing the reality of learning participants’ activities. With an average fitness almost above the average as well as a strong behavioral and structural appropriateness, the WF nets are to be enriched in fixing either the process flow or structure filling out the gap where needed in order to give a complete representation of the learning behavior in FLOSS. Nevertheless, we succinctly explain some of the deviations observed in terms of process flow on these repositories and their inherent performance differences.

In verifying the Novice role, our results indicate that the analysis produced a fitness value \( f(\text{Novice}_{\text{Net}}, \text{Novice}_{\text{Log}}) = 0.52500534 \) for the bug reporting repository. This is above average and also considering that these repositories generated thousands of unique traces given their volumes, a full fitness was not really warranted. Likewise, the comparison of the Novice WF Petri net on the source code and reviews repositories produced respectively \( f(\text{Novice}_{\text{Net}}, \text{Novice}_{\text{Log}}) = 0.4751042 \) and \( f(\text{Novice}_{\text{Net}}, \text{Novice}_{\text{Log}}) = 0.43482816 \). These close values on both repositories do not fare away from the bug reports data set and are indicative of the fact that the initial models (WF Petri nets) would require to be enriched for about 60% of what the real behavior exhibits in Event Logs to be able to provide a more realistic representation of the process flow at this level. This discrepancy can be noted in the diagnostic models as seen in the Figures 263-265.

These figures perpetuate almost the same trend on some aspects as already discussed in the previous phases. As the fitness goes above the average, we notice that sometimes this can be attributed largely to missing, invisible or consumed tokens generated during analysis and there are no direct restrictions with regard to the precedence relationships. However, as the fitness drops under 0.5, the general trend is that a lot of activities are required to follow a specific order in light of other activities as they are executed together or form part of the same traces.

Looking at Figure 263, we observe the diagnostics for the Novice behavior on bug reporting and one can notice that no constraints in terms of precedence relations are identified with a fitness value of 0.52500534. Almost the same observation can be made on the reviews repository as illustrated in Figure 264 with only a single constraint. This specifies that the activity Submit Bug Report in the commit state never follows Post Questions in the review state when they are part of a trace affecting the fitness value to 0.43482816. This implies that every time a user had to submit a bug report while also posting questions they had to post questions first before a bug report was submitted. Furthermore, Figure 265 shows quite a considerable number of activities with precedence constraints on source code. Some activities never follow others in certain traces while they always follow some others in different instances on source code.
Chapter 9: Validation of Learning Process Models: Conformance Checking

Figure 263. Diagnostics Model appropriateness on Bug Reports for Novice in FLOSS during Maturation phase

Figure 264. Diagnostics Model appropriateness on Reviews for Novice in FLOSS during Maturation phase
Figure 265. Diagnostics Model appropriateness on Source code for Novice in FLOSS during Maturation phase
As this will be expressed in the final Petri net and given the number of these constraints, we make use of two distinct activities to exemplify the existence of these two constraints. Figure 265 specifies that the activity Submit Source Code in the commit state always follows Modify Source Code in the same state when executed in the same flow. Every time a user modifies some code, they would have to resubmit the code or rather submit the modified code. On the other hand, Submit Bug report or Submit Documentation in the commit state never follows Reply To Suggestion in the develop state. When a Novice executes a number of tasks that include these two activities, he/she would never execute any of these two activities after Replying to Suggestions.

The values for precision or behavioral appropriateness regarding these three repositories are $a_B (Novice_Net, Novice_log) = 1.0$, $a_B (Novice_Net, Novice_log) = 0.99593496$ and $a_B (Novice_Net, Novice_log) = 0.63986015$ for the analysis respectively on bug reports, reviews and source code repositories. As hinted, one can notice an almost perfect score on both bug reports and reviews in terms of behavioral appropriateness influenced largely by the lack of precedence relation constraints as pointed out. It is expected that the behavior on source code deviates a little bit given the presence of a number of constraints. A score of 0.63986015 indicates that over the period of time we considered for data analysis, the WF Petri net represents about 64 % of the real behavior for the Novice at this phase. Looking at the most frequent paths and activities as well as common trends between these three repositories’ Event Logs, we thus produce a final Petri net like in the rest of the cases reflecting and incorporating all constraints where needed and more importantly providing a more descriptive learning process for the Novice during this phase.

Finally, the results for the structural appropriateness observed in the same repositories are given by values of $a_S (Novice_Net, Novice_log) = 0.7037037$ and $a_S (Novice_Net, Novice_log) = 0.8888889$ for both bug reports and reviews repositories while $a_S (Novice_Net, Novice_log) = 0.7407407$ and $a_S (Novice_Net, Novice_log) = 1.0$ are structural appropriateness values for the source code repository. The perfect match on the last data set is explained by the presence of all identified activities while the first two do not include the activity Submit Documentation as they contain data solely about tasks related to either bugs or just general reviews on application artifacts as previously described.

These results are reflected in the final Petri nets for all the respective repositories. In Figures 266, 269 and 273, a Petri net representation of the control-flow is given according to the behavior as captured through the corresponding logs in bug reports, reviews and source code.

In Figure 266, the Novice performs a number of activities on Bug reports and has about 4 possible paths or directions to follow as learning activities traces. The first path starts with PostQuestion before choosing to either WriteSourceCode and then SubmitBugReport before executing the last block of activities or the Novice can at this point choose between two more options: AnalyzeThreadProgression or ReplyToSuggestion before executing any of activities including ReviewCommentContents, ReviewPosts or PostCommentOnCode. After this part is executed, the Novice will either SubmitBugReport as a result or proceed to any of the last block of activities.
Figure 266. Final Learning Model for Novice during Maturation phase on Bug reports with a replay of 1000 traces

Figure 267. First Grouping of Activities indicating first point of iteration
Figure 268. Second Grouping of Activities indicating second point of iteration
Chapter 9: Validation of Learning Process Models: Conformance Checking

Figure 269. Final Learning Model for Novice during Maturation phase on Reviews with a replay of 1000 traces

Figure 270. First grouping of activities depicting possible starting points of paths
Chapter 9: Validation of Learning Process Models: Conformance Checking

Figure 271. Second grouping of activities depicting possible points of iterations

Figure 272. Third grouping of activities depicting possible points of iterations
Figure 273. Final Learning Model for Novice during Maturation phase on Source Code with a replay of 1000 traces

Figure 274. First grouping of activities depicting possible points of iterations and beginning of paths
Figure 275. Second grouping of activities depicting remaining possible points of iterations and traces paths
From the reviews repository, as seen in Figure 269, the Novice follows almost the same pattern but in this instance, the activity SubmitBugReport comes before activity ReplyToSuggestion which precedes in turn a group of 4 activities which can be followed distinctively including AnalyzeSourceCode, PostCommentOnCode, ReportBugs, and GiveSuggestion. From this point, the path can continue in a number of different directions by either executing ProvideFeedback or PostCommentOnCode or AnalyzeThreadProgression or SubmitCode and complete the phase. Alternatively, the path can be completed by ModifySourceCode, FixBugs then ReviewSourceCode and finally ReportBugs if any. A minimum level of iteration is can be observed at any point during this execution.

Lastly, Figure 273 depicts the same Petri net for the Novice behavior on the source code’s repository. The general trend is mostly the same as in the previous two nets but the main differences of course lay in the presence of much longer traces. That is we can find here, a lot of activities being executed successively forming a longer trail than in the two previous repositories. It is helpful that the general trend is observed in three different repositories. We have managed to somehow give an exclusive idea of what the learning representation can be from each individual repository but at the end of this chapter, we conclude by providing the final and definite Petri nets resulting from the fusion between all of these distinct data sets.

We ran the same experiments for the role of Expert in this process flow. Using the same repositories, we noted almost the same pattern as the Novice’s behavior. However, with the absence of the activity Submit Documentation on the source code repository for the Expert, all three data sets exhibit practically the same behavior as shown by the same values obtained for the structural appropriateness. Furthermore, this can also be seen with the fitness value on source code which almost exceeds the average 0.5.

These values are given as follows. The fitness for the Expert Log on bug reports, reviews and source code is given by the values \( f(\text{Expert Net, Expert Log}) = 0.5210679 \), \( f(\text{Expert Net, Expert Log}) = 0.40341926 \), and \( f(\text{Expert Net, Expert Log}) = 0.51980984 \) respectively. We observe that the Expert WF net on two of the repositories fits at an average degree while it goes a little below with reviews.

Figures 276-278 present the graphical diagnostics as expressed with these values as well as the appropriateness we discuss in a subsequent paragraph. Once again we can notice in Figure 276 that the fitness value obtained for the bug reports attests to the fact that a considerable number of instances are analyzed when it comes to bug reporting and a lot of activities come into play but our WF Petri net does represent at some acceptable degree the process flow and precedence rules of activities as recorded in the Event Log. Also, just like was the case with the Novice, the behavior of the Expert does not fare far always as seen in Figure 277 in reviews. Two constraints are picked and identify that activity Send Feedback in the analyze state always follow activities Analyze Source Code and Comment On Code in the develop state. This is plainly self-explanatory and alludes to the idea that every time a source code is analyzed, when the Expert puts relevant comments, this translates into sending appropriate feedback.

However, things are not as straightforward for the source code repository where a number of activities, almost all of them excluding Analyze source code in the analyze state, have precedence constraints as the spaghetti-looking diagnostic model in Figure 278 demonstrates. We look at two distinct examples as an illustration as the final Petri net will be reflective of all these properties. The activity Review Code in the commit state always follows many other activities including Review Report in the same phase. This implies that when these two activities appear in a trace, the process execution warrants a review of a source code relating to a report after the latter has been reviewed to probably verify its correctness. The last example is the activity Send Feedback in the commit state never follows Run Source Code in the develop state but the opposite is possible when executed together in a process flow.
Chapter 9: Validation of Learning Process Models: Conformance Checking

Figure 276. Diagnostics Model appropriateness on Bug reports for Expert in FLOSS during Maturation phase

Figure 277. Diagnostics Model appropriateness on Reviews for Expert in FLOSS during Maturation phase
Figure 278. Diagnostics Model appropriateness on Source Code for Expert in FLOSS during Maturation phase
As it pertains to precision or behavioral appropriateness, the results indicate that on bug reports and reviews, the standard description from the WF Petri net explains the behavior respectively at values $a_B (Expert_Net, Expert_log) = 1.0$ indicating a perfect match between the two artifacts and $a_B (Expert_Net, Expert_log) = 0.992$ clearly almost providing the same indications. However, the disparity expressed through the spaghetti diagnostic model in Figure 278 produces a value $a_B (Expert_Net, Expert_log) = 0.13308084$ for the source code.

Finally, the results for the structural appropriateness observed in all three repositories surprisingly for the source_code as well are given by values of $a_S (Expert_Net, Expert_log) = 0.6060606$ and $a_S (Expert_Net, Expert_log) = 0.8947368$ respectively for the simple and advanced structural appropriateness. This suggests that the structure of our WF net fairly represents the general perception of the Expert learning behavior.

Just like in all the previous cases, we captured all these results in the form of Petri nets from each repository to represent the learning behavior exhibited by the Expert. All the activities executed draw a picture of long traces or paths that the Expert executes during the course of the learning process. Figures 279, 282 and 285 give depict authentic Petri nets as extracted from these repositories. We segmented them accordingly to demonstrate points of iterations and grouping of activities.

Certainly, as expected, some differences in the occurrence and flow of execution for certain activities as suggested by the fitness and appropriateness measures but the general trend remain quite similar. This means that most of the same behavior for the Expert can be found on each distinct repository. In the next section, we conclude by merging the Event Logs opened from these repositories in order to obtain a consolidated Petri net that is representative of the learning behavior for the Expert in this last phase of the learning phase.
Chapter 9: Validation of Learning Process Models: Conformance Checking

Figure 279. Final Learning Model for Expert during Maturation phase on Bug reports with a replay of 1000 traces

Figure 280. First grouping of activities depicting possible starting point of traces and iteration points
Figure 281. Second grouping of activities depicting possible second points of iteration
Figure 282. Final Learning Model for Expert during Maturation phase on Reviews with a replay of 1000 traces
Figure 283. First grouping of activities showing possible starting points for distinct traces and points of iterations
Figure 284. Second grouping of activities showing possible points of iterations and continuation of paths
Figure 285. Final Learning Model for Expert during Maturation phase on Source Code with a replay of 1000 traces

Figure 286. First grouping of activities showing possible starting points for distinct traces and points of iterations
Figure 287. Second grouping of activities showing possible points of iterations and continuation of paths
9.7 Models Comparison: Learning Process Models in Floss Environments

In this chapter, the main purpose entailed performing a step by step and detailed analysis of our initial a-priori models that depict the learning behavior in light of the real captured occurrence from Event Logs. Through conformance analysis, we have established at some extent that the a-priori models give a generic representation of learning process from a simplistic perspective while the Event Logs provide a much richer and inclusive approach in describing learning processes as they happen. Nevertheless, it is crucial to have a final representation of these learning patterns based on the obtained Event Logs that supports these observations. This can be a visual and graphical depiction of the Learning process models in FLOSS.

However, before we produce a final and inclusive representation of the learning behavior in FLOSS, it is important, as a reminder, to justify this choice with the adoption of multiple repositories upon which we conducted experiments during the analysis. In every phase, we conducted experiments on different datasets that present some particularities in terms of the level of performance during the replay on a-priori models. Given the dynamism of such an endeavor, all the insights from these Event Logs have contributed significantly in exploring and studying learning processes in FLOSS. Therefore, we propose to merge these logs for the same participant in order to consolidate and produce a harmonized representation of the Learning processes. In addition to this, it is important that we compare these global inclusive models from logs to the a-priori models in order to clearly establish and demonstrate the differences and similarities or lack thereof between them as proved by the conformance analysis.

Surely, the conformance analysis has helped relate these two set of models at least formally and semantically through traces replay. The detailed analysis provided in the previous sections embodies the many points of correlation between these artifacts. Nevertheless, as a closing argument, we think it is important to summarize and provide an abstract interpretation of this comparison by looking at the final models and the a-priori models. This can be achieved by representing the two set of artifacts in the same format. This can be accomplished by solely working on the representation files (.pnml) that contain Petri net – like, and compatible, formal representations for both sets of models.

Therefore, the objectives in this last part of the chapter are threefold. Primarily, perform a merger of the Event Logs from the different repositories in order to produce models with a consolidated and inclusive representation of learning behavior in FLOSS. Secondly, ensure that both the de jure and de facto models are equally represented. Lastly, clearly describe and provide a high-level interpretation of the comparison between these two sets of models.

In order to perform the merger, we make use of the Artificial Immune System algorithms [145] to merge the logs and reanalyze them in order to produce global models that inclusively describe participants’ learning behavior in FLOSS. The core of the AIS algorithms consist in automatically executing two tasks in order to merge two Event Logs together. First, traces from both Event Logs that belong to the same process execution are linked together, and then they are merged into one trace to be stored in a new file [145].

Locating these traces in both Event Logs that can be put together is premised on a couple of factors just like in an artificial immune system. The algorithm steps are influenced by a fitness function, which determines the quality of discovered parts of the solution [145]. Results and experiments from implementing this algorithm show that a set of factors are defined in order to indicate if traces from the two Event Logs belong together and this is used to calculate the fitness function score. Furthermore, the sum of all factors has to lead the algorithm to an optimal solution. Details and a description of these immune system inspired factors can be found in the work presented in [145]. The AIS algorithms are implemented in a plug-in available in the ProM Framework. In our work, we make use of this plug-in to
combine Event Logs that describe the same process executions. We define the settings in such a way that unique traces from each log are kept and most importantly, ensure that we do not repeat identical traces, that is, if two traces are identical from the two logs, only one is kept. Hence, we combine the Event Logs from both Mailing archives and Internet relay chats for both Novice and Expert in the Initiation and Progression phases. We also, merge the Event Logs from bug reports, reviews and source code for the corresponding participants. As a result, we produce 6 Event Logs for 6 global process models as classified according to the learning phases below.

The second objective is to make sure that both the *de jure* and *de facto* models are portrayed in the same graphical representation. This is very important as it ensures that the interpretation of these models is done from the same perspective. In order to accomplish this, we perform a simulation with each model and perform a replay of 1000 traces on these models. This is based on the work by Rogge-Solti *et al.* [146-149] on capturing the performance of a system or business process as accurately as possible. The plug-in produced as a result of these studies can be used to enrich existing Petri net models with stochastical information obtained from a log [149]. Using the information from our Event Logs from which the final models were obtained, some stochastic information is retrieved and on the basis of this simulations are performed on all our sets of models accordingly. The purpose is to simulate runs through a Petri net model and store the simulated traces in a log. Another motivation for us in adopting this technique was its ability to allow for models to be able to capture the behavior of the environment and resources as accurately as possible [146]. This objective is achieved through algorithms that are based on the notion of alignments and have been implemented as a plug-in in the process mining framework ProM [146] that we make use of.

Lastly, in order to pair the two sets of models, we segment long models where possible in order to capture major points of convergence or divergence between the models. The resultant segments are set apart as unique figures and capture both activities and the frequency of process execution as simulated. We perform this analysis gradually by looking at models in each of the three phases of the learning process.
9.7.1 Initiation Phase

In this phase, we have models representing the learning behavior for both the Novice and Expert. Figure 288 is the final Petri net depiction in process model for the Novice based on the merged Event Logs. A replay of 1000 traces is captured in order to explain the process execution. On the other hand, Figure 291 represents the Initial a-priori model for the Novice with 1000 traces replay in order to equally illustrate the process execution and get enough information to compare the two models.

A technique we have adopted is to break down the model where there are groups of activities that either execute alternatively or represent potential points of iteration or starting and ending points in the traces. Figures 289, 290 and 292 exemplify the implantation of the technique. We have used the same approach in explaining individual model for every separate repository in the previous sections but with fewer details.

On analyzing the Novice’s behavior, we recall the figures that the conformance analysis produced with regards to the two main metrics we set, fitness and appropriateness. The replay of the Event Logs on the a-priori model produced a fitness of $f = 0.045625925$ on Mailing archives and $f = 0.3585199$ on Internet Relay chat messages. We certainly established that these figures were largely influenced by the size of the logs from both repositories as they contained hundreds of thousands of traces.

However, in considering Figures 289 and 292, representing respectively the real behavior Novice as proved from the Event Logs and the theorized behavior depicted in the a-priori model, there are three major characteristics that need to be highlighted. Some of these highlights and conclusions can also be drawn from the rest of the models in the next phases as we describe later.

- Firstly, a general remark is the presence of iteration points in the final model and the lack thereof in the a-priori model. The initial model presents a simplistic sketch of how learning behavior is perceived to occur based on observations and interviews while the final models provide a much richer representation of the learning behavior with all possible points of iterations as well as any variations in the behavior one can anticipate for the Novice in this phase.

This can be observed in Figures 289, 290 and 292 zooming on these groupings of activities that incarnate these variations. The a-priori model shows a simple trace of activities such as $\text{FormulateQuestion} \rightarrow \text{IdentifyExpert} \rightarrow$ and then three equal alternative options are provided where the Novice can either $\text{PostQuestion}$, $\text{PostMessage}$ or $\text{CommentPost}$ as seen in Figure 292. And then the trace continues with $\text{ContactExpert} \rightarrow \text{SendDetailedRequest}$ marking the end of the trace. This shows the possibility of 3 distinct full traces that the a-priori model provides with no iteration.
Chapter 9: Validation of Learning Process Models: Conformance Checking

NOVICE

Figure 288. FINAL PROCESS MODEL for NOVICE from merged Event Logs

Figure 289. First Grouping of activities explaining iterations and starting points of traces
Figure 290. Second Grouping of activities explaining iterations and ending points of traces
Chapter 9: Validation of Learning Process Models: Conformance Checking

Figure 291. INITIAL APRIORI MODEL for NOVICE with a replay of 1000 traces

Figure 292. Grouping of activities explaining alternative activities in the traces
On the other hand, the real behavior captured as seen in Figure 288 caters for iterations and variations in the traces. The model offers two possible starting points for traces. The first possibility as followed by 496 of the replay traces depicts a starting point and then a choice between either *IdentifyExpert, PostMessage* or *CommentPost* and then complete the phase. This is in advent of short traces and one can find cases where a Novice performs all of these three activities alternatively a number of times. The second possibility, which largely represents the Novice’s behavior starts from *FormulateQuestion → PostQuestion* and then either execute *ContactExpert → SendDetailedRequest* which triggers a series of iterations. Alternatively, the Novice can proceed to the last set of activities and choose either *IdentifyExpert, PostMessage* or *CommentPost* and complete the phase. This process execution in the final model offers the possibility of 24 distinct traces that the Novice can execute.

From the description above, one can notice that the final model submerges the process execution of traces found in the a-priori model. The only difference lies in the execution of activity *IdentifyExpert* initially assumed to be executed after *FormulateQuestion* but that the real behavior has discovered to be expressed as a distinct and alternative activity that can be executed in lieu of *PostMessage* or *CommentPost*. Instead, the activity *PostQuestion* is found to be executed immediately after *FormulateQuestion* which makes more sense as when a question is posed, this implies that it has just been formulated.

This slight difference would explain the discrepancy in the two models expressed through the fitness metric. In replaying the Event Logs on the a-priori model, a lot of times there will be a lot of missing and remaining tasks because the activities *IdentifyExpert* and *PostQuestion* are found in different places.

Nevertheless, it is possible that instances of all three distinct traces found on the a-priori model be found in the final model Event Logs, hence the values for the fitness metric on both the Mailing archives and IRC chats repositories were not \( f = 0 \). Furthermore, values obtained for appropriateness express submersion. The behavioral appropriateness for both Event Logs on the a-priori model obtained indicated respectively \( a_B = 0.8492753 \) and \( a_B = 0.6624337 \) while the structural appropriateness value was \( a_S = 0.6666667 \) for both repositories. This simply shows that the structure of the initial a-priori model can claim to represent the real behavior at 67% given the discrepancy explained. The final model hence provides a complete and more accurate representation of how the first learning activities in a learning process are executed by the Novice.

Our experiments, therefore, were critical in providing a complete view and more enriched representation of the Novice behavior. We therefore follow the same pattern of analysis to give a global view on the role of the Expert.

The models representing the learning behavior for the Expert are found in Figures 293 and 296. Figure 293 is the final Petri net depiction in process model for the Expert based on the merged Event Log depicting an animation or replay of 1000 traces that illustrates the process execution. The second model depicted in Figure 296 represents the Initial a-priori model for the Expert with 1000 traces replay in order to equally illustrate the process execution and get enough information to compare the two models. Therefore, two main observations can be made from these models:

Firstly, we can also notice that the real Expert behavior represented in the final model makes provisions for both iterations and possible traces variations in a more complete fashion than does the a-priori model. The points of iterations and possible variations in traces are expressed by
means of activities grouping presented in Figures 294 and 295 for the real behavior final model and Figure 297 for the a-priori model.

Looking at these figures, especially Figures 296 and 297 for the a-priori model we can notice that the model offers solely 3 possibilities of distinct traces with no iterations. The first trace is ReadMessages→ContactNovice→CommentPost/SendFeedback. The second trace is ReadSourceCode→ContactNovice→CommentPost/SendFeedback and finally the last trace would be ReadPost→CommentPost→ContactNovice→CommentPost/SendFeedback.

On the other hand, Figures 294 and 295 complete this initial representation with missing patterns: iterations and variations. One can observe that the real behavior identifies the possibility of having 18 distinct traces. The first traces are distinct execution of three activities as seen in Figure 294 including ContactNovice, CommentPost/SendFeedback and ReadSourceCode. The second set of traces as seen in Figure 295 include ReadMessages→ReadPost→CommentPost and then either iterate in the same sequence the number of times required and then execute any of the last three alternative activities ContactNovice, CommentPost/SendFeedback and ReadSourceCode or simply proceed to these activities without iterating. A total of 18 distinct traces can be observed in this model.

The differences observed in the sequencing of some paths have an impact on the fitness as well as appropriateness values as obtained from the conformance analysis. In the real behavior final model, one can observe that activities such as ContactNovice and CommentPost/SendFeedback do not follow each other but are rather distinct activities that can be executed alternatively. Also, the sequence ReadPost→CommentPost→ContactNovice from a trace in the a-priori model can be completely executed in the final model. While the other two traces are possible parts of the traces in the final model with the exception of the ContactNovice activity being a missing task in these instances.

These differences certainly impact on the final results for the fitness respectively $f = 0.58433735$ on Mailing archives and $f = 0.5337318$ on IRC chats messages. Nevertheless, the second metric demonstrate that the three traces from the a-priori model can be entirely found on the final model with a structural appropriateness value of $a'_S = 1.0$. Equally, the conformance results for the behavioral appropriateness indicate that the a-priori model is submerged in the final model with values of $a'_B = 1.0$ on Mailing archives and $a'_B = 0.8333333$ on IRC chats.

Therefore, the general trend that our approach has observed in tracing the learning activities in this first phase is that the final models provide a more inclusive and complete representation of the participants’ behavior with missing characteristics expressed in the a-priori models. In the next section we look at the second and subsequently third phases of the learning process.
EXPERT

Figure 293. FINAL PROCESS MODEL for EXPERT from merged Event Logs
Figure 294. First Grouping of activities explaining iterations and starting points of traces for Expert

Figure 295. Second Grouping of activities explaining iterations and ending points of traces for Expert
Figure 296. INITIAL APRIORI MODEL for EXPERT with a replay of 1000 traces

Figure 297. Grouping of activities explaining alternative activities in the traces for the EXPERT
9.7.2 Progression Phase

In this phase, the focus is on the medium activities in the learning process that the Novice executes. Gradually, the Novice can interact more with peers, seeking answers to any arising questions along the way. Therefore, the number of activities increases for both the Novice and Expert. We look at these progressively just like in the previous phase starting with the Novice.

To understand the interplay between the a-priori model and the final model for the Novice, we consider the representation learning behavior depicted in the respective Figures 29 and 301. Figure 298 represents the real behavior of Novice as captured from the merged final Event Log while Figures 299 and 300 depict the grouping of activities that explain points of variations in the traces. Similarly, Figure 301 shows the a-priori model for the Novice in this second phase of learning. The grouping of activities can be seen in Figure 302. Looking at these figures, one can make the following observations:

- Firstly, the continuation of the trend observed in the first phase with both participants. The a-priori model is quite simple and presents no possibilities for iterations. The sole variation is the options for two branching of activities for traces in Figure 302. In total, only 2 possible unique traces are provided in this de jure model. The first part shows quite a linear trace that includes ProvideFeedback → PostQuestions → ReplyPostedQuestions → PostQuestions → SendFeedback → AnalyseSourceCode before completing the traces with either ReportBugs or CommentOnCode.

The final model in Figure 299 on the other hand also does not present any possibility of iteration but shows two groupings of activities that demonstrate variations generating a total of 96 possible unique traces. Figures 299 and 300 depict the starting point for the traces with two distinct beginning. Firstly, the Novice can start with PostQuestions and then move to the second block of alternative activities in Figure 299 where some or all activities could also be skipped before ending with any of the activities in the second and last grouping of activities that offer the same possibilities of either executing alternative activities or skipping them accordingly. Secondly, the Novice has the choice to skip the first activity PostQuestions and proceed to the second and third block of activities as specified in the first case. Hence, 2 (starting possibilities) * 8 (variations offered in the first block of activities) * 6 (final variations offered in the second block of activities) total 96 possible unique traces.

- Equally, we deduce that the wide difference in terms of possible unique full traces between the two sets of models surely has a bearing on the conformance results. At this point, it is safe to assume that the same trend is observable in the rest of the models even in the next phase. This is true for all the models considered in this analysis.

The conformance analysis demonstrates that the Event Logs on both Mailing archives and IRC chats play out on the a-priori model respectively with a fitness of $f = 0.15142721$ and $f = 0.10818714$. When instances of 96 possible full traces are played on the model that only caters for two unique traces, one can understand these values that explain the discrepancy in terms of fitness.

- Similarly, the results for the second metric also show the presence of parts of the two unique traces in the a-priori model on the final Event Log. With values of $a_S = 0.7777778$ for the structural appropriateness on both repositories and $a_B = 0.64597225$ on Mailing archives and $a_B = 0.69343656$ for the Internet Relay chat messages, it is quite clear that the a-priori model does not cater for all possibilities of traces even in terms of structure that the final model represents.
Figure 298. FINAL PROCESS MODEL for NOVICE from merged Event Logs during Progression
Figure 299. First Grouping of activities explaining variation points for Novice during Progression

Figure 300. Second Grouping of activities explaining variation points for Novice during progression
Figure 301. INITIAL APRIORI MODEL for NOVICE during Progression with a replay of 1000 traces

Figure 302. Grouping of activities on the A-PRIORI MODEL depicting variations for NOVICE with a replay of 1000 traces
On analyzing the Expert’s learning behavior in this phase, we look at the Figures 303 and 307 respectively illustrating the final Petri net as a result of the Event Logs merger and the initial a-priori model. The grouping representing possible iterations and variations of traces are illustrated in Figures 304, 305 and 306 for the final model while Figures 308 and 309 group variations in traces for the a-priori model. We draw the following conclusions expressed in these figures:

- The a-priori model contains four separate branching of activities that provide variations in producing 16 distinct traces. In Figure 308, 2 first possibilities are observed between ReviewThreadCode and ReviewThreadPost before each of these two culminates in executing SendReply as a result of these reviews before executing either PostQuestions or ReplyPostedQuestions. After this first grouping, a sequence of activities that each unique trace will follow includes SendFeedback→ReportBugs→ReplyToPost. Then comes the last block of activities that depicts two separate branching. The first branching includes choosing between AnalyzeSourceCode and RunSourceCode and the last two options includes ReportBugs or CommentOnCode.

The analysis from the final model in Figure 303 does include points of iteration as expressed in Figure 304 as well as two more grouping of activities that provide unique alternative execution of activities in Figures 305 and 306. As seen in Figure 304, two alternatives can start the process. The Novice can start with CommentOnCode→AnalyzeSourceCode→RunSourceCode that can either be iterated before proceeding to the second block of activities passing through executing or skipping activity SendReply or the Novice can skip the first block of activities and directly execute SendReply or skip it and execute any of the second block activities. The second block of activities depicted in Figure 305 gives options to execute either of ReportBugs, ReplyPostedQuestions or PostQuestions. This leads to the execution of the last block of activities including ReportBugs, ReplyToPost, SendFeedback, ReviewThreadcode and ReviewThreadPosts. Hence, 3 (2 starting points + iteration of the second point) * 2 (variations in choosing either SendReply or skipping it) * 6 (possibilities for executing or skipping second block of activities) * 10 (variations for executing or skipping any of the 5 activities in the last block of activities) will give a total of 360 possible unique traces.

- The results of the conformance analysis exhibit this trend with values for fitness being respectively \( f = 0.26572186 \) and \( f = 0.1161989 \) for Mailing archives and IRC chat messages. Again in this case, it is predictable that when instances of 360 possible full traces are played on a model that only caters for 16 unique traces, the fitness values will be very low like in this case.

Similarly, the results for the second metric can also explain that a model with 16 distinct traces can present both the structural and behavioral appropriateness values respectively \( a_S = 0.5 \) and \( a_B = 0.93826085 \) for Mailing archives and \( a_B = 0.37608695 \) for IRC chat messages.
EXPERT

Figure 303. FINAL PROCESS MODEL for EXPERT from merged Event Logs during Progression

Figure 304. First Grouping of activities explaining variation points for Expert during progression
Figure 305. Second Grouping of activities explaining variation points for Expert during Progression

Figure 306. Third Grouping of activities explaining variation points for Expert during progression
Figure 307. INITIAL APRIORI MODEL for EXPERT during Progression with a replay of 1000 traces
Figure 308. First grouping of activities on the A-PRIORI MODEL depicting variations for EXPERT with a replay of 1000 traces

Figure 309. Second grouping of activities on the A-PRIORI MODEL depicting variations for EXPERT with a replay of 1000 traces
9.7.3. Maturation Phase

In this Maturation phase, the Novice has significantly improved the skills learned in the community and moves towards sharing this knowledge. The Expert role at this point consists more of peer monitoring and knowledge sharing. At this point even the Expert learns through interactions that take place in the community. Also, the number of activities grows significantly from just 8 activities in the second phase to 18 in the maturation phase. The experiments conducted in this phase are interpreted according to the participants, starting with the Novice and concluding with the Expert.

The \textit{de jure} and \textit{de facto} models for the Novice are represented in Figures 310 and 313 respectively. Figures 311 and 312 are two main groupings of activities that depict the critical points of iterations and variations in traces on the final model while Figures 314, 315 and 316 present the same for the initial a-priori. Looking at these figures, one can make the following observations:

- The a-priori model provides three potential groupings of activities that introduce variations in the paths and traces the Novice execute during this phase. The first grouping in Figure 314 shows the starting of two traces through either \texttt{AnalyzeSourceCode} or \texttt{AnalyzeDiscussions} \rightarrow \texttt{AnalyzeThreadProgression} before deciding on three distinct activities including \texttt{SubmitCode}, \texttt{SubmitDocumentation} and \texttt{SubmitBugReport}. The second grouping in Figure 312 gives options to equally choose between \texttt{ModifySourceCode}, \texttt{PostCommentOnCode}, \texttt{WriteSourceCode}, \texttt{FixBugs}, \texttt{GiveSuggestion} and \texttt{ReplyToSuggestion}. Lastly, the last grouping in Figure 316 gives 2 sets of branching respectively between \texttt{ProvideFeedback} and \texttt{PostQuestions} and then execute either \texttt{ReviewSourceCode} \rightarrow \texttt{ReportBugs} or \texttt{ReviewPosts} \rightarrow \texttt{ReviewCommentContents}. The total number of possible full unique traces can be calculated by counting 2 (starting points) \* 3 (first branching) \* 6 \* 2 \* 2 which equals 144 traces.

Equally, the final model can be interpreted by observing the two main groupings in Figures 311 and 312. In Figure 311, one can observe two possible starting points for traces. Firstly, the Novice can \texttt{SubmitDocumentation} before executing the activities and variations in the second grouping or they can start with executing two successive subgroups of activities with the first including activities such as \texttt{SubmitBugReport}, \texttt{ReplyToSuggestion}, \texttt{WriteSourceCode} and \texttt{PostQuestions} resulting in another branching with activities \texttt{AnalyzeSourceCode}, \texttt{ReviewCommentContents}, \texttt{ReviewPost} and \texttt{GiveSuggestion}. The next step is to either execute \texttt{AnalyzeThreadProgression} or move to the last block of activities.

The last block of activities entails executing either of activities that include \texttt{AnalyzeDiscussions}, \texttt{SubmitCode}, \texttt{ProvideFeedback}, \texttt{ModifySourceCode} \rightarrow \texttt{FixBugs} and then either execute or skip \texttt{PostCommentOnCode} \rightarrow \texttt{ReviewSourceCode} \rightarrow \texttt{ReportBugs}. The second block of activities can be iterated. These groupings and variations produce a total number of unique traces that can be computed by taking (1 (first starting point) + (8*8*2 (from two subgroup in the second starting point))) \* 32 (which can be iterated) \* 32 obtaining 132096 possible unique and complete traces.

- The conformance analysis demonstrates that the replay of the Event Logs from source code, reviews and bug reports on the a-priori model respectively produced fitness values $f = 0.4751042$, $f = 0.43482816$ and $f = 0.52500534$. We can thus deduce that some traces and their instances from these logs could be replayed in full on the model explaining the differences in these values. The second metric, appropriateness, suggests however that the behavior of the a-priori model can be representative of the real behavior with values of $a_a = 1.0$, $a_b = 0.99593496$ and $a'_b = 0.63986015$ in that order for the Event Logs on bug reports, reviews and bug reports with regards to behavioral appropriateness. The values of the structural appropriateness appear to also be high for all three repositories respectively $a_S = 0.8888889$ , for bug reports and reviews and $a'_S = 1.0$ for source code. The slight differences in these values have been justified and motivated in Section 9.6.2 when we dealt with each repository individually.
NOVICE

Figure 310. FINAL PROCESS MODEL for NOVICE from merged Event Logs during Maturation
Figure 311. First Grouping of activities explaining variation points for Novice during Maturation

Figure 312. Second Grouping of activities explaining variation points for Novice during Maturation
Figure 313. INITIAL APRIORI MODEL for NOVICE during Maturation with a replay of 1000 traces

Figure 314. First grouping of activities on the A-PRIORI MODEL depicting variations for NOVICE with a replay of 1000 traces
Figure 315. Second grouping of activities on the A-PRIORI MODEL depicting variations for NOVICE with a replay of 1000 traces

Figure 316. Third grouping of activities on the A-PRIORI MODEL depicting variations for NOVICE with a replay of 1000 traces
The last sets of models both *de jure* and the *de facto* models capture the behavior of the Expert in this last phase. Figures 317 and 321 depict the main model for the Expert, respectively the final and initial a-priori model. Both of them include points of variations that depict either iterations or simply differentiate between alternative executions of certain activities. These remarks are to be noted from the analysis:

- The a-priori model provides four potential groupings of activities that introduce variations in the paths and traces the Expert executes during this phase. Firstly, two starting points for traces are shown in Figure 322, which shows that the Expert would either execute AnalyzeSourceCode or AnalyzeDiscussions→AnalyzeThreadProgression and then execute SendFeedback before deciding on three distinct activities from the second grouping including ReviewCode, ReviewDocumentation and ReviewReport. After this grouping, the next activity is SendFeedback again and then either RunSourceCode→ReportBugs or AnalyzeSourceCode→CommentOnCode before executing either ReviewThreadCode or ReviewThreadPosts as seen in Figure 323. The resulting activity is SendReply and then execute the last block of activities as seen in Figure 324 through executing either ReviewPosts→ProvideFeedback or ReviewSourceCode→ReportBugs which complete this phase for the Expert. This produces a total of possible unique traces to 48 (2 (2 starting points)*3(first 3 branching) *2 (as seen in third grouping)*2 (from second branching in third grouping) * 2 (from last grouping)).

The final model on the other hand, can be interpreted by looking at the three main groupings in Figures 318, 319 and 320. In Figure 318, one can observe two possible starting points for traces. Starting from ReviewThreadPosts and either going directly to the second block of activities or executing SendReply and then iterate. The second set of activities indicates that the Expert can execute or skip either of the activities or sequence thereof including ReviewThreadCode and move to the last block of activities, AnalyzeDiscussions and then execute SendFeedback or move to the last block of activities, ReportBugs→RunSourceCode and then execute or skip either SendFeedback or AnalyzeSourceCode. The next step would be to either execute or skip ReviewCode. The last option is about executing a subgroup of three short sequences that iterate. These include ReviewSourceCode→ReportBug, AnalyzeSourceCode→CommentOnCode, ReviewPosts→ProvideFeedback and finally AnalyzeThreadProgression. Next, the activity ReviewReport can either be executed or skipped. These groupings and variations produce a total number of unique traces that can be computed as follows. Considering 4 (from grouping 1) *28 (second grouping)* 19 (third grouping) producing 2128 possible unique and complete traces.

- The conformance analysis demonstrates that the replay of the Event Logs from source code, reviews and bug reports on the a-priori model respectively produced fitness values \( f = 0.51980984, f = 0.40341926 \) and \( f = 0.5210679 \). We can thus deduce that some traces and their instances from these logs could be replayed in full on the model explaining the differences in these values. The second metric, appropriateness suggests however that the behavior of the a-priori model can be representative of the real behavior with values of \( a_B = 1.0, a_B = 0.992 \) and \( a_B = 0.13308084 \) in that order for the Event Logs on bug reports, reviews and bug reports with regards to behavioral appropriateness. The value of the structural appropriateness appears to also be high enough reaching \( a_S = 0.8947368 \) for all three repositories. The slight differences in these values have been justified and motivated in a prior section when we dealt with each repository individually.
EXPERT

Figure 317. FINAL PROCESS MODEL for EXPERT from merged Event Logs during Maturation
Chapter 9: Validation of Learning Process Models: Conformance Checking

Figure 318. First Grouping of activities explaining variation points for EXPERT during Maturation

Figure 319. Second Grouping of activities explaining variation points for EXPERT during Maturation
Figure 320. Third Grouping of activities explaining variation points for EXPERT during Maturation
Figure 321. INITIAL APRIORI MODEL for EXPERT during Maturation with a replay of 1000 traces

Figure 322. First grouping of activities on the A-PRIORI MODEL depicting variations for EXPERT with a replay of 1000 traces
Figure 323. Second grouping of activities on the A-PRIORI MODEL depicting variations for EXPERT with a replay of 1000 traces

Figure 324. Third grouping of activities on the A-PRIORI MODEL depicting variations for EXPERT with a replay of 1000 traces
Figure 325. Fourth grouping of activities on the A-PRIORI MODEL depicting variations for EXPERT with a replay of 1000 traces
9.8 Conclusion

In this chapter, we have established a comprehensive global picture of learning processes in FLOSS. First through conformance analysis as well as models comparison, our experiments have provided indication that learning processes as expressed through the a-priori models are incomplete. We performed a conformance analysis and the discrepancies detected can be explained by the presence of factors such as:

— The assumption that values for the metrics are obtained based on the replay of available traces in the logs. These logs are dynamic and cannot be predicted.
— The presence of possible short traces resulting in increased missed and artificial tasks during replay.
— The total number of possible unique traces from a-priori models in all three phases is exponentially less than the actual number of possible unique traces from final models as seen in Table 2.
— These discrepancies demonstrate that the a-priori models are incomplete and submerged in the final models either completely or for the most parts of traces.
— Therefore the final models provide a more complete representation of learning processes in FLOSS.

Furthermore, the final models were obtained through merging the Event Logs from the different repositories to obtain consolidated and inclusive models. We are confident this representation can be reproduced following our approach. To express the completeness of the final models, we depict the number of unique possible traces in Table 16. The way these figures were obtained has been explained in the previous section for each case.

Table 16. Projection of Unique Traces in both A-Priori and Final Models

<table>
<thead>
<tr>
<th>NUMBER OF POSSIBLE UNIQUE TRACES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase</td>
</tr>
<tr>
<td>Initiation</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Progression</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Maturation</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Furthermore, it is important as a final remark that we relate some observations from Learning Analytics (LA) to the findings of the conformance analysis. The purpose of this action is twofold. First, we aim to provide more support and systematically substantiate our findings within the realm of Learning Analytics. Second, it is critical as a final remark, to contextualize the differences between a-priori and final (posteriori) models as shown in Table 16 in light of the initial determinations in Chapter 3, Section 3.3.

Learning Analytics (LA), as a new analytics discipline, is defined as the measurement, collection, analysis, and reporting of data about learners and their contexts, for the purposes of understanding and optimizing learning and the environments in which it occurs [184]. Making use of data
collected from learning management systems or any educational technology, experts can predict students’ behavior and provide useful insights that can help improve both the contents of courses and their delivery mode. Although this is a new discipline attracting interest from Big data and data science community, some observations appeal to our analysis and can explain some of our findings. After proposing a comprehensive and largely validated Framework for Learning Analytics, Greller and Drachsler [180] highlight 2 important things that can be observed in our findings.

First, although numerous techniques from well-established disciplines such as machine learning and statistical inference have been used on educational data in structured environments such as online learning classrooms, there is little evidence for the support of constructivist approaches to learning, where learning is seen as an active cognitive process in which learners construct their own concepts of the world around them [180]. In Learning Analytics, this is inferred indirectly by looking simply at the students’ grades [180].

In our case, this can be observed in the differences between the number of unique traces in a-priori and final models as shown in Table 16. These discrepancies attest that it is not possible to neither infer the complete behavior of learners in FLOSS communities nor accurately estimate the number of total traces FLOSS members make during the learning process. Nevertheless, given that FLOSS environments are not primarily learning systems, it is a remarkable finding that process maps as well as the related activities captured in Chapter 6-8 provide a clear indication of the presence of learning in these ecosystems.

Secondly, any pedagogical value in the context of Learning Analytics can only be observed as manifested and represented by the available data [180,182-184]. This means that we can only see learning values through the data. The results of the conformance analysis reflect this observation. The underlying thought is that the extent of learning is well understood in the context of the data we analyze. Different data sets may give different indications of learning as knowledge and experiences vary considerably among Novices and Experts. We have amply and contextually described the differences in terms of fitness and appropriateness of Event Logs for both the Novice and Expert across the 3 phases. Nevertheless, this observation provides additional insights into potential reasons why we observe such differences. Especially, with conformance analysis, the more data we have, the more unique traces we generate, and they determine the results we obtain.

Furthermore, a number of differences between the Novice and Expert can be interpreted and understood based on this observation. As detailed in Sections 9.1 and 9.2, during conformance, we execute a replay of all the traces from the Event Log on the a-priori model and the final results are an accumulation of fitness or appropriateness values for each trace. It is possible that with a different set of Event Logs, a different behavior executed by the Novice or Expert, the values for fitness or appropriateness would differ.

For example in Table 16, we notice that the a-priori models for both the Novice and Expert are estimated to generate solely 3 traces during the initiation phase, based on the considerations detailed in Chapter 3 and Section 3.3. We note that the final or a-posteriori models produce respectively 24 and 18 traces. We can deduct from these differences that the Novice’s Event Log is more likely to have a lesser fitness value than the Expert’s because of the differences in the number of traces. The more traces we have in an Event Log, the more variations the learning behavior present. This is observed through the values for both fitness and appropriateness as shown in Table 15. The Novice’s and Expert’s Event Log respectively record a fitness value $f = 0.045625925$ and $f$
= 0.58433735 as well as advanced structural appropriateness $a_S = 0.6666667$ and $a_S = 1.0$. The same pattern can be observed in the rest of phases.

We therefore assert that our initial structuring of a-priori models still hold based on the considerations we described. To our knowledge, there is no definitive framework that determines the sequence in which learning activities are expected to occur in FLOSS environments. Our initial estimates constitute a comprehensive model upon which the analysis was based and the final results do not entirely annihilate them. Given the dynamic and sporadic nature of FLOSS environments, we make the observation that it is not possible to have an accurate determination of when and how an action will occur at all times. Even if we incorporated iterations in the a-priori models, it is less likely that the log replay will yield a fitness value of 1.0. The obvious reason is that, it has been proved that in Learning Analytics [180], the pedagogical value of learning is observed in the available data.

Nevertheless, the a-priori models provided enough proof that without empirical analysis of FLOSS data as provided in this thesis, it would be difficult to estimate and understand the presence and dynamics of learning in FLOSS environments.
CHAPTER 10: CONCLUSION AND FUTURE WORK

10.1 Introduction

Over recent years, FLOSS communities have consistently raised significant research interests from different perspectives on a number of questions [1-11, 57-68]. One of the critical motivations behind this surge of interests is the quality of software applications or FLOSS products whose popularity shows tremendous growth. FLOSS communities have produced products such as operating systems (i.e. Linux), network services (i.e. Apache), high-end applications (i.e. MySQL, PostgreSQL, Sendmail), Content Management Systems (i.e. Drupal) and Learning Management Systems (LMS), such as Moodle [1, 11].

One of the critical perspectives from these interests in researching on FLOSS communities pertains to learning. FLOSS environments have been tipped to provide learning opportunities for participants [6-11, 27-30]. Furthermore, a number of institutions of higher education have conducted pilot studies to assess the implementation and inclusion of FLOSS projects participation in traditional settings of learning [6, 10, 28-36]. Such initiatives are driven by the need to give students an opportunity to acquire practical software engineering experience through involvement in real and non-fictitious projects. Students are also given an opportunity to foster their ability to work in collaborative environments while solving real problems. While such studies in FLOSS have been widely investigated [1-2, 23] and [28-36], the resulting observations have in most part or almost entirely been based on data collected using surveys and questionnaires or through reports from observers who have been part of the community for a defined period of time [23]. Therefore, one of the key motivations for our study has been to provide empirical support to such findings and also determine to what extent existing visual perceptions of learning behavior could be traced from real data and learners’ behavior.

Providing empirical evidence from FLOSS data presupposes, mining the corresponding repositories in order to retrieve relevant information and construct models that exemplify a representation of FLOSS users’ learning behavior in these environments. A myriad of studies have attempted to develop and implement mining techniques for the analysis of these repositories [57-73]. In Chapter 2, we presented the main techniques and tools used to mine software repositories from FLOSS environments. In spite of these studies that are regularly presented in conferences and workshops as part of the annual Mining Software Repositories (MSR) series, there exists no or limited effort for empirically mining learning processes in FLOSS.

Therefore, in this thesis, the work was also driven by the need to foster the understanding of learning patterns generated by knowledge acquisition mechanisms occurring in FLOSS environments and provide an empirical representation of such patterns. In order to accomplish this, we made use of Process Mining. Following a defined methodology, our approach proposed and implemented a way to extract data records from FLOSS repositories, analyze the data and construct Event Logs needed for analysis.
In this last chapter, we briefly summarize, firstly how the methodology defined in Chapter 1 led to the accomplishment of our set objectives and hypotheses, secondly, define a way forward in light of the obtained results.

10.2 Research Objectives vs. Research Output

At the beginning of our study and within the context of our approach, we set to verify and confirm the existence of learning processes in FLOSS communities from users’ audit trails captured in Event Logs, provide insights into the ways in which these processes occur (process discovery), and contextualize these findings in light of a number of normative or a-priori models.

Therefore, we confidently highlight that we showed the culmination of our empirical results for the presence of learning activities in FLOSS and the extent to which they form part of discussions and exchanges in this ecosystem from chapters 5-9. Each of the chosen repositories to analyze for tracing learning activities according to the corresponding learning phase has provided significant insights to support our hypotheses. We thus relate these findings to the originally defined objectives as follows:

[obj1] Develop and implement new techniques that can help extract semantically information related to participants’ activities during learning processes

This objective has been achieved through the adoption of Semantic Search for information extraction. As described in Chapter 4, we proposed the use of a set of catalogs that define major key phases that can help identify related activities and ultimately extract participants’ activities in FLOSS repositories. The pseudocode described in Chapter 5, provides the proposed step by step approach in making use of these catalogs in order to efficiently mine data from such repositories.

[obj2] Apply these techniques to generate semantic-based logs

With the use of Semantic Search, coupled with the catalogs, we produced Event Logs that are based on the semantic content of messages in Openstack’s Mailing archives, IRC messages, Reviews. Bug reports and Source code to retrieve the corresponding activities. Considering these repositories in light of the three learning process phases, we produced an Event Log for each participant (Novice or Expert) in every phase on the corresponding dataset. Therefore, we first produced 14 Event Logs that we detailed accordingly in Chapters 5-9; then we derived final merged Event Logs from different repositories containing data about the same participant in the same corresponding phase resulting in 6 global and inclusive Event Logs in Chapter 9.

[obj3] Process mine these logs to produce related process models

Our attempts to producing process models start with depicting process maps to track the actual behaviour as it occurs in Openstack repositories (Chapters 5-8), before concluding with final Petri net models representative of learning processes in FLOSS as a result of conformance analysis and models comparison in Chapter 9. In Chapters 6-8, we visually represented the behaviour of Novice and Expert based on the Event Logs built in the previous chapters. For every dataset in the corresponding learning phase, we produced 3 process maps depicting the overall learning behaviour for every FLOSS member irrespective of whether they are Novice or Expert before breaking down with a representation for each respective participant. In total, we produced 21 process maps, which
in essence are a representation of process models on real data in Chapters 6-8; we also generated 14 process models in the form of Petri nets for every participant on each dataset in Chapter 9 and 6 more Petri net models depicting final learning process models in Chapter 9.

Evaluate the models and discuss the results

This last objective was achieved, first briefly in Chapters 6-8 where we contextualized the value of discovered process maps in light of related statistical indications. Moreover, this was also accomplished in Chapter 9 where we conducted conformance analysis and compared the obtained models to the a-priori models. We conclude that the tentative Workflow nets introduced in Chapter 3 represent a more simplistic representation of learning processes in FLOSS. This is because our experiments with Event Logs from Openstack repositories have demonstrated that the real behavior are more complete and submerge these simplistic models.

10.3 Experiments output versus Hypotheses

The attainment of these objectives has provided enough evidence and data needed for us to verify our hypotheses defined in Section 1.4 of Chapter 1. After developing and applying a new approach based on Process Mining data extracted from FLOSS repositories (Mailing archives, IRC messages, Reviews, Bug reports and Source Code) available on the Openstack platform, we have demonstrated in this thesis that we can trace participants’ activities and message exchange in FLOSS communities and construct process models that can be validated for conformance or learning patterns discrepancy with existing approaches as depicted in Figure 11. This approach was based on these three hypotheses that can be verified as follows:

[H1] FLOSS communities are indeed possible learning environments

This hypothesis has been confirmed first through literature studies reported in [6-11, 27-36]. Furthermore, the statistical data about the participation of FLOSS users on process maps speaks volumes of the existence learning processes in these environments.

The Mailing archives dataset has proved largely to be an adequate environment to identify all activities pertaining to either making contacts or beginning of potential collaboration. It has proved extensively that in FLOSS, learning processes start at great scale. From a total of 54762 emails exchanged on 15 mailing lists, it was possible to identify 14 learning activities executed in 565 cases over 123401 events. The Novice performs more of these Initiation phase activities with 99.54% of all the activities, while the Expert accounts for only about 0.36% of these events.

Considering this description, one can observe practically the same trend on IRC messages as on Mailing archives. Learning activities do occur at a significant rate during messages exchange on these forums and the results from mining IRC messages provide insights on the starting activities that trigger these processes. While the dataset boasts over 5 million messages, only 2142690 of these were good enough for our analysis. It has emerged that 14 of the learning activities could be identified through 605965 events across 28 cases where the Novice, accounts for the biggest share of traffic during these exchange. The Novice executes the activities about 86.53% of the total number compared to the Expert who operates the related activities for 7.22%.
Equally, during the Progression phase, the evidence produced from this mining experiment solidifies the finding in terms of the existence of learning processes in FLOSS as well as the scale at which they occur. Primarily with Mailing archives, the total number of events produced amounts to 117014 with the Novice still performing the most activities at 95.89% while the Expert increased from 0.36 to 3.63% of the total learning activities. Secondly, on IRC messages, 739578 events are produced over 28 cases where the Novice contributes 45.95% of the total number of events, while the Expert performs up to 38.06%.

Finally, during Maturation phase, the choice of the three datasets largely explains how learning occurs at this stage. Both the Bug reports and the Source code demonstrate the commitment of the Novice to seek answers and interact as much as possible in strengthening the acquired skills. With a participation of 49.22% for the Novice against 46.72% for the Expert and 46.19% against 42.04% respectively on Bug reports and Source code, the Novice still engages significantly in learning. On the last dataset, Reviews, we notice an increase in the Expert’s role. The Expert performs activities to the tune of 40.36% of total number of activities against 22.17% for the Novice. This significant increase in the involvement of the Expert supports our conclusion about the Expert being more involved with reviewing Novice’s work.

[H2] FLOSS repositories can be analyzed using Process Mining tools and techniques for learning patterns identification and representation

In Chapter 9, we produced final learning processes as a result of analyzing Mailing archives, IRC messages, Reviews, Source Code and Bug reports from Openstack. As part of this analysis, we got to understand that a-priori models presented an incomplete representation of learning processes in FLOSS communities. Therefore, the analysis generated enough details to represent the real learning behavior exhibited in FLOSS according to repositories we analyzed. The experiment demonstrated that the total number of possible unique traces from a-priori models in all three phases is exponentially less than the actual number of possible unique traces from final models. A summary of these differences can be seen in Table 17 and 18 and plotted in Figures 326 and 327 respectively for Novice and Expert.

![Figure 326. Depiction of differences in Number of Unique traces between A-priori and Final](image)
Table 17. A Summary of Differences in Number of Unique Traces between A-Priori and Final Models for Novice

<table>
<thead>
<tr>
<th>Phase</th>
<th>A-Priori Model</th>
<th>Final Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiation</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>Progression</td>
<td>2</td>
<td>96</td>
</tr>
<tr>
<td>Maturation</td>
<td>144</td>
<td>132096</td>
</tr>
</tbody>
</table>

Figure 327. Depiction of differences in Number of Unique traces between A-priori and Final

Table 18. A Summary of Differences in Number of Unique Traces between A-Priori and Final Models for Expert

<table>
<thead>
<tr>
<th>Phase</th>
<th>A-Priori Model</th>
<th>Final Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiation</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>Progression</td>
<td>16</td>
<td>360</td>
</tr>
<tr>
<td>Maturation</td>
<td>48</td>
<td>2128</td>
</tr>
</tbody>
</table>

[H3] New methods and supporting systems (Algorithms) for learning process identification in FLOSS can be developed and evaluated.

From the process maps to the final process models obtained and then evaluated in Chapter 9, we can verify this hypothesis. The empirical data we have produced demonstrate that our approach...
consisting of the use of Semantic Search and key phrases catalogs can produce help identify learning processes from FLOSS repositories.

10.4 Process Mining FLOSS repositories: Research Contributions

The contributions of our study to the body of knowledge have been extrapolated throughout this thesis. With regards to the obtained results, we can point out that the significance of this study is manifold.

— Firstly, we highlight our contribution to the vast research area of Mining Software Repositories (MSR) as guided by the MSR series. In addition to providing empirical evidence for the existence of learning processes in FLOSS, our approach can be replicated to model process analysis for further studies such as collaboration formations, software component evolution, software component task allocation and management etc.

— Secondly, our study has shown how we have successfully assessed and implemented Process Mining with FLOSS repositories. Process Mining is a fairly young field that is gaining momentum and promises to grow in the data science and information analysis community. Its adoption and application on FLOSS repositories contributes significantly in advocating its potential as well as the invaluable benefits that can be derived from it.

— Thirdly, this study provides more insights and supporting evidence to all institutions of higher education that plan to introduce participation in FLOSS projects as an alternative to providing solid software engineering skills to CS students.

— Lastly, this work can serve as a starting point for the design of a new learning model and accompanying learning tools that can offer practical experience and the opportunity to develop an array of skills in a holistic learning approach.

10.5 Future work

In addition to the highlighted values added from this study, certainly the results can pave ways for further studies in order to foster the understanding of these learning processes and the reasons they occur in a given fashion. Therefore, we hope to replicate this approach to conduct similar experiments on different repositories from different FLOSS platforms such as SourceForge, Freshmeat, and GitHub provided the data is available. These further studies will certain provide more insights on learning processes from a different perspective.

Furthermore, these learning processes can still be studied further in order to answer more questions such as the ability to predict the occurrence of certain activity flows given an attribute, the ability to predict users’ duration or stay in a FLOSS project that is more likely to yield learning processes.

Many additional studies, especially regarding process analytics, can still be conducted in FLOSS environments following the novelty offered by our approach. With a properly defined ontology, Semantic Search or text mining can be used to identify the activities from textual data in FLOSS repositories in order to reconstruct real behavior recorded in data through Process Mining techniques.
REFERENCES


[34] Fernandes, S., Cerone, A., & Barbosa, L. A preliminary analysis of learning awareness in FLOSS projects.


Mukerjee, K., Porter, T., & Gherman, S. (2011). Semantic Mining of Documents in a Relational Database. In KDIR (pp. 146-158).


