A Study of Schema & Software Co-evolution for Relational Databases in Free Open-Source Projects

A Thesis

submitted to the designated
by the Assembly
of the Department of Computer Science and Engineering
Examination Committee

by

Fation Shehaj

in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE IN DATA AND COMPUTER SYSTEMS ENGINEERING

WITH SPECIALIZATION
IN ADVANCED COMPUTER SYSTEMS

University of Ioannina
School of Engineering
Ioannina 2021

Examining Committee:

- **Panos Vassiliadis**, Professor, Department of Computer Science and Engineering, University of Ioannina (Supervisor)
- **Nikolaos Mamoulis**, Associate Professor, Department of Computer Science and Engineering, University of Ioannina
- **Apostolos Zarras**, Associate Professor, Department of Computer Science and Engineering, University of Ioannina

DEDICATION

To my family.

ACKNOWLEDGEMENTS

First of all, I would like to offer my special thanks to Professor Panos Vassiliadis, my research supervisor, for his precious guidance and the collaboration we had. Moreover, I wish to thank my parents for all the support and encouragement they offered me throughout my study.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABSTRACT

Fation Shehaj, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, June 2021
A Study of Schema & Software Co-evolution for Relational Databases in Free Open-Source Projects
Advisor: Panos Vassiliadis, Professor

In this dissertation, we attempt to study and make an analysis on the co-evolution of the database schema and source code. Studying the co-evolution is especially important as it can identify patterns on how the code development can impact the schema evolution, with the purpose to help designers and developers spend less time modifying the storage and processing system for the provided information. Also, through this study, the potential effects on software maintenance that would emerge from the dependence of the source code and the database schema can be reduced or even improve the performance of the software and potentially the development time. The key question of this thesis is: Is there a correlation on how the evolution of a software source code affects the evolution of the database schema, and if so, can we categorize them? To answer this question, we used data from the commit history of three hundred and fifty (350) projects, collected with the help of the GitHub platform. The projects were divided into six categories, these categories are: frozen, almost frozen, focused shot & frozen, moderate, focused shot & low and active.
First, we made an extensive, manually (non-automated) analysis of six randomly selected projects. We tried to understand how the code and the database schema evolve simultaneously and find possible patterns. We also developed software that uses the data files from GitHub to visualize the code and database changes in real-time using bar charts to help us identify possible patterns.
Finally, to draw more derailed conclusions, as we are not interested in all types of changes in a SQL file, but only those that affect the schema of the database, we used files that contained more information about the history of changes in the database, to export diagrams with the cumulative changes for each program. This tool,

incorporated in the previous one, is made to the standards of the Heraclitus tool (GitHub: HeraclitusFire) and has also the ability to export these graphs into a web format to better summarize the information.

ΕΚΤΕΤΑΜΕΝΗ ΠΕΡΙΛΗΨΗ

Φατιόν Σέχαϊ, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, Ιούνιος 2021
Μελέτη της Συν-εξέλιξη του Σχήματος και του Κώδικα για Σχεσιακές Βάσεις Δεδομένων σε Ανοιχτού Κώδικα Έργα.
Επιβλέπων: Παναγιώτης Βασιλειάδης, Καθηγητής

Σε αυτή την διπλωματική εργασία επιχειρούμε να κάνουμε μια μελέτη και ανάλυση στην συν-εξέλιξη του σχήματος της βάσης δεδομένων και του πηγαίου κώδικα. Η μελέτη αυτής της συν-εξέλιξης είναι ιδιαίτερα σημαντική καθώς μπορεί να εντοπίσει μοτίβα στον τρόπο με τον οποίο η ανάπτυξη του κώδικα μπορεί να επηρεάσει την εξέλιξη του σχήματος, με σκοπό να βοηθήσει τους σχεδιαστές και προγραμματιστές να αφιερώνουν λιγότερο χρόνο για την τροποποίηση του τρόπου αποθήκευσης και επεξεργασίας της παρεχόμενης πληροφορίας. Επίσης, μέσω αυτής της μελέτης μπορούν να μειωθούν οι πιθανές συνέπειες στην συντήρηση του λογισμικού που θα προκύπταν λόγο της εξάρτηση του κώδικα και του σχήματος της βάσης δεδομένων ή ακόμα και να βελτιωθεί η απόδοση του λογισμικού και ενδεχομένως του χρόνου ανάπτυξης του.
Το βασικό ερώτημα αυτής της διπλωματικής εργασίας είναι: Υπάρχει κάποια συσχέτιση στον τρόπο που η εξέλιξη του πηγαίου κώδικα ενός λογισμικού επηρεάζει την εξέλιξη της βάσης δεδομένων, και αν ναι, μπορούμε να τα κατηγοριοποιήσουμε; Για να απαντήσουμε την συγκεκριμένη ερώτηση, χρησιμοποιήσαμε δεδομένα από το ιστορικό των αλλαγών από τριακόσια πενήντα (350) προγράμματα (projects), που συλλέχθηκαν με την βοήθεια της πλατφόρμας του GitHub. Τα έργα (προγράμματα) χωρίστηκαν σε έξι κατηγορίες, συγκεκριμένα: frozen, almost frozen, focused shot & frozen, moderate, focused shot & low and active.
Αρχικά, κάνουμε μια εκτεταμένη, μη αυτοματοποιημένη ανάλυση σε έξι τυχαία επιλεγμένα προγράμματα, προσπαθώντας να καταλάβουμε τον τρόπο που εξελίσσεται ο κώδικας και το σχήμα βάσης ταυτόχρονα και να βρούμε πιθανά μοτίβα. Επίσης, αναπτύξαμε ένα λογισμικό που χρησιμοποιεί τα αρχεία με το

ιστορικό δεδομένων από το GitHub, για να οπτικοποιήσουμε σε διαγράμματα μπάρας τις αλλαγές στον κώδικα και τη βάση στον χρόνο, ώστε να μας βοηθήσει να εντοπίσουμε πιθανά μοτίβα.

Στην συνέχεια, για να εξάγουμε πιο λεπτομερή συμπεράσματα, καθώς δεν μας ενδιαφέρουν όλες οι αλλαγές σε ένα SQL αρχείο, αλλά μόνο όσες επηρεάζουν το σχήμα της βάσης, χρησιμοποιήσαμε αρχεία που περιείχαν περισσότερη πληροφορία για το ιστορικό αλλαγών στην βάση δεδομένων, για να εξάγουμε διαγράμματα με τις σωρευτικές αλλαγές για κάθε πρόγραμμα. Το εργαλείο αυτό, ενσωματώθηκε στο προηγούμενο, έγινε στα πρότυπα του εργαλείου Ηράκλειτος (GitHub: HeraclitusFire) και έχει την δυνατότητα να εξάγει τα γραφήματα αυτά σε διαδικτυακή μορφή για καλύτερη σύνοψη της πληροφορίας.

CHAPTER 1

INTRODUCTION

1.1 Goals

The life cycle of each product includes a series of changes, there is no doubt about that. A software product is not an exception to that maintenance process. The reasons for those changes usually aim to fix potential problems and faults or extend the product's features. Almost every software product consists of a database. Due to the increase of functionalities the source code usually becomes more dependent on the database. This entails a sequence of changes and modifications to the database, usually causing schema changes. The terms schema and software co-evolution refer to those changes.

So far, there is a limited number of studies on this topic. That indicates the difficulty of analyzing the schema and software co-evolution due to the unavailability of a large number of open-source projects, with a database and the history in the correct form, without gaps, to allow us to establish a solid conclusion. The importance of studying the schema and software co-evolution can be realized if we consider the problems that can be occurred due to the software changes, without the proper database changes, leading to failures, information loss or even retrieving the wrong information. To help the product maintenance or database evolution, it is critical to identify potential patterns. In this way, we can eliminate all these effects caused by

the source code and database dependence, and possibly reduce the time and effort required or even optimize the product.

Our approach to the topic consists of the research and the tools created to assist it. First, we tried to better understand how schema and software co-evolve doing a manual analysis of six randomly selected projects from our data. After that, we automate the process of analyzing the history activity of software and schema. We created a new tool that we named Evolution Chart Exporter, using the HeraclitusFire (on GitHub) as a reference for the line and bar chart exporters. With this tool, we were able to visualize the changes in each commit to the source code and database and also visualize the cumulative activity for the project and the schema changes over time.

In our research, we decided to answer two main questions.

- ***Research Question 1:*** What percentage of the projects demonstrates a "hand-in-hand" co-evolution, where the schema evolution heartbeat closely follows the heartbeat of the project?
- ***Research Question 2:*** What percentage of projects demonstrates the 80-20 rule reported in the literature [3], i.e., 80% of the schema evolution activity was obtained in the first 20% of the time?

In Chapter 4 we analyze and answer these two questions. We present the process and the algorithms we used to reach these findings.

1.2 Structure of the Thesis

This thesis consists of o four chapters. The contents of each chapter can be summarized as follows. In Chapter 2, we analyze and highlight some of the most significant work, done to contribute to the topic of schema co-evolution with code and we explain how our work differentiates from the others. In Chapter 3, we made a manual analysis on schema and code co-evolution for six randomly selected projects. We also created a tool to help us visualize the number of files changed in each commit for the project's life. In this chapter, we introduced six questions and we tried to answer them. In Chapter 4, we created a tool to compute the cumulative activity for each project using files with more information on what changed in each schema commit.

CHAPTER 2

RELATED WORK

This chapter presents the research work that has been previously done in schema and software co-evolution and what has been achieved by the efforts in the literature. The interest in this field has been extremely small in the last decades. In the first subsection of this Chapter, we review the previous efforts and report on their results. In the second subsection, we demonstrate a brief comparison of our work and the studies of the first section.

2.1 Case Studies of Schema and Software Co-Evolution

In 2009, Dien-Yen Lin and Iulian Neamtiu [1] focused their research on the collateral evolution of applications and databases. The authors use the term *collateral evolution* to designate the lack of consistency when database and application code do not coexist in sync. In this context, the authors define a formula for collateral evolution. To understand the formula, we will define some parameters the authors used. First, the authors used D to denote the data and $F(D)$ to denote the data format. Furthermore, the authors used $F_C(D,X)$ to denote the expected format by client C, version X and $F_S(D,Y)$ to denote the format provided by the server S, version Y.

Now we can introduce the formula authors defined, let X and Y be the data client and server versions that result from collateral evolution. Let $F_C(D,X)$ be the format expected by client C and let $F_S(D,Y)$ be the format provided by the server S. The collateral evolution is potentially incorrect if $F_C(D,X) \neq F_S(D,Y)$. The authors used two open-source projects, Mozilla and Monotone to study co-evolution and identify changes to database schemas. Next, the authors studied the evolution of data format in three major database management systems, SQLite, MySQL and PostgreSQL. The main findings of this study are condensed as follows:

- The most frequent modifications are database schema changes followed by additions and deletions of tables or attributes.
- Concerning the problem of data and software co-evolution, the database schema and source code does not always evolve in sync. To avoid conflicts with database and source code, Mozilla uses two methods, the first mechanism is to ignore the collateral problem and assume that if a database exists, then the schema version and the schema version of the app are in sync. The second solution is to determine the versions of the application and database, perform the schema migration and then access the database. On the other hand, in Monotone, the authors encounter the collateral evolution problem with the use of a centralized routine. The authors investigated table additions and deletions and found "orphan" and useless tables that take up space.
- A different problem that the authors investigate is the file format that a database management system produces. DBMS producers often change the file format from one version to another for reasons like performance or storage size. The most common way for dealing with problems that may show up is to dump the database into a batch file of SQL commands and recreate the database with the use of the new DBMS.

In 2011, Shengfeng Wu and Iulian Neamtiu [2] presented their work on schema evolution analysis for embedded databases and proposed a system to automatically extract embedded database schemas and source code with the purpose to automatically compute the schema evolution. The authors studied the evolution

within eighteen years of four popular applications containing embedded databases. The key findings of their study are outlined in the list below.

- A high frequency of table and attribute deletions denotes that embedded databases are more prone to restructuring, rather than continuous growth.
- The early stages in schemas tend to have a higher number of changes, while the later versions include few changes and the database stabilizes over time.
- The embedded databases have a lower change rate than the enterprise-class databases.

In 2013, Dong Qiu, Bixin Li and Zhendong Su [3] made an empirical analysis for the co-evolution of schema and code in database applications. The authors used ten popular open-source projects for their study and posed three research questions to answer how schemas and code co-evolve. These are:

- ***How frequently and extensively do database schemas evolve?*** This question helps to understand whether they intensively evolve during an app's development and maintenance process.
- ***How do database schemas evolve?*** This question helps to understand what schema change types usually occur in practice.
- ***How much application code has co-changed with a schema change?*** This question helps to understand the real impact on application code. We are also

interested in whether certain schema change types tend to have more impact on code than others.

The steps authors used to extract the information from project repositories and answer these questions can be synopsized as follows:

- ***Locate schema files.*** Extract the schema files, most files have the .sql suffix although some projects specify schema information using embedded SQL statements (e.g. PHP files).
- ***Extract DB revisions.*** Identify DB revisions (commits) that contain modifications to schema files, if a schema file is among the changed files of revision i, then i is a DB revision.
- ***Extract valid DB revisions.*** Filter those revisions containing only related schema changes.
- ***Extract atomic changes.*** Authors extract all schema changes by manually comparing schema files of contiguous valid DB revisions.
- ***Co-change analysis.*** Analyze and calculate the real impact that has been triggered by these atomic schema changes.

The authors analyzed how information is present in evolution history. Suppose R is the set of valid DB revisions and C_r represent all committed changes in the r revision, r is the current one under analysis. Then SC_r are the schema changes and $CC_r$are the code changes, both are subsets of the C_r. The RC_r is the actual code changes

caused by SC_r. CC_r and RC_r both can be empty, on the contrary to the SC_r. The authors introduce four possible co-change situations:

- (S1) $CC_r = \emptyset$ and $RC_r \neq \emptyset$.
- (S2) $CC_r = \emptyset$ and $RC_r = \emptyset$.
- (S3) $CC_r \neq \emptyset$ and $CC_r \cap RC_r \neq \emptyset$.
- (S4) $CC_r \neq \emptyset$ and $CC_r \cap RC_r = \emptyset$.

The findings of this process are outlined in the subsequent list:

- Database schemas evolve at a high rate during their lifecycle, on average 90 atomic schema changes per year. Also, the variety of their changes follows a similar distribution in all ten projects.
- In most of the projects, their schema size approaches 80% of their maximum value within the first 20% of their lifetimes.
- Schema changes urge considerable code modifications. Some change types trigger more code changes than others.
- More schema changes happened in a small number of tables and nearly half of schema tables did not change.
- Additions of tables or columns and datatype changes are the most frequent changes at the low – level of change categories.
- Co-change analyses can be crucial to automate database application evolution. Moreover, the authors suggest three functionalities that a tool like this should have.

2.2 Comparison to the State of the Art

In the previous section, we tried to present the most relevant work made until now in the schema and software co-evolution. We attempted to give a synopsis of the contribution of each work, leading us to a better understanding of the mechanism

that determines how schema and software co-evolve. In our work we analyze this mechanism in a big dataset, using 350 projects and their history, we tried to understand how both, schema and code co-evolve and locate patterns. We also made the first steps to automate the process creating tools to visualize the history activity for each project.

CHAPTER 3

MANUAL ANALYSIS OF SCHEMA AND CODE CO-EVOLUTION

In this chapter, for a better understanding of schema and source code co-evolution, we choose to make a deeper analysis of six randomly selected projects and manually examined the history of the commits for these projects. We will present to you our findings, the schema changes and the triggered code changes. Finally, we will cite our findings from our deep analysis of the six projects and we will explain how this study helped us further with our study. We are going to present you also a tool we created to examine visually the occurrence of code and schema changes over time, we named that tool EvolutionChartExporter (ECE). Finally, we tried to locate patterns from the projects.

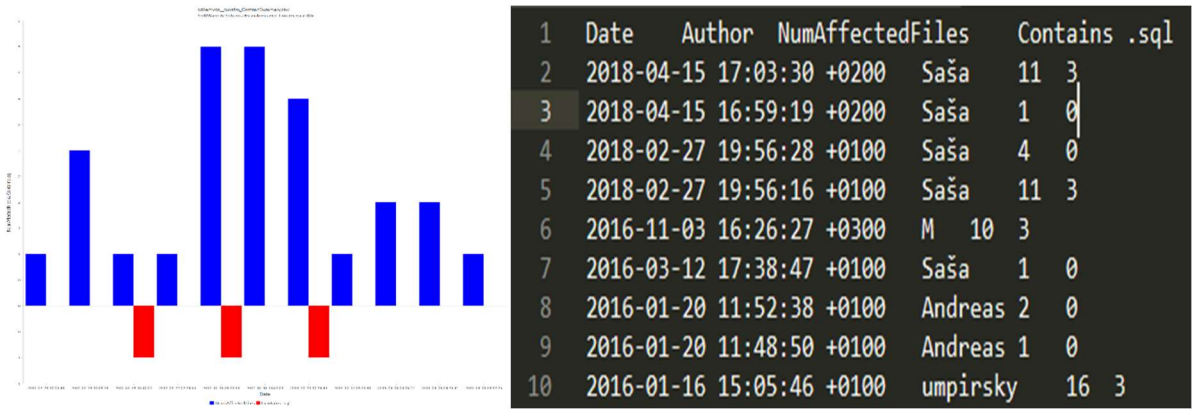
3.1 EvolutionChartExporter

3.1.1 Introduction to EvolutionChartExporter

The main reason we decided to create the EvolutionChartExporter tool is to visualize how much code and schema changes have been committed over time. To create the EvolutionChartExporter tool, we used as reference the chart exporter source code from HeraclitusFire. Firstly, we will make a brief explanation of how this tool works. In the next subsection, we will deeply analyze the EvolutionChartExporter, what are its imports and how we extracted these files needed and what are the exports of this tool. We will also introduce you to the design of the EvolutionChartExporter. In the section that will follow, we will mention the tests we made to evaluate this tool. As we mentioned, we collected the commit history for 350 projects from GitHub. Having this amount of data is unable to select manually which projects we are going to investigate further. Using EvolutionChartExporter, we are able to have a quick view of each project's commit history.

3.1.2 How it works, its Architecture and Design

As we mentioned, we used HeraclitusFire as a base to create our tool. HeraclitusFire has the ability to create different types of charts. For our needs, we used the bar exporter. The specific thing about this chart exporter is the ability to plot bars above and below the x-axis. We used the x-axis for the time and the y axis for the number of changed files in a commit. Above the x-axis, we plotted the number of changed source code files and below the x-axis, we plotted the number of .sql changed files. The y-axis counts the changes made. Figure 3.1 below shows an example: (a) the exported image from the EvolutionChartExporter and (b) the format of the input file.



	Date	Author	NumAffectedFiles	Contains .sql
2	2018-04-15 17:03:30 +0200	Saša	11	3
3	2018-04-15 16:59:19 +0200	Saša	1	0
4	2018-02-27 19:56:28 +0100	Saša	4	0
5	2018-02-27 19:56:16 +0100	Saša	11	3
6	2016-11-03 16:26:27 +0300	M	10	3
7	2016-03-12 17:38:47 +0100	Saša	1	0
8	2016-01-20 11:52:38 +0100	Andreas	2	0
9	2016-01-20 11:48:50 +0100	Andreas	1	0
10	2016-01-16 15:05:46 +0100	umpirsky	16	3

(a) (b)

Figure 3.1 Bar chart exported from EvolutionChartExporter (a) the exported image from the EvolutionChartExporter and (b) the format of the input file

For our study, we separate the 350 projects into six taxa. The taxa we created were the following, Frozen, Almost Frozen, Focused Shot and Frozen, Moderate, Focused Shot and Low and Active. The algorithm used to classify the 350 projects was firstly introduced in [4] and is shown in Figure 3.2.

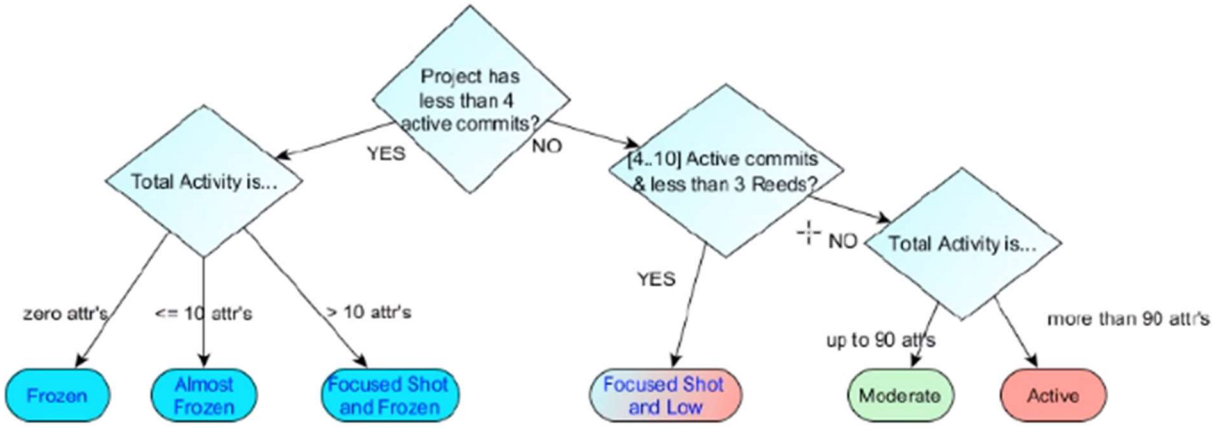


Figure 3.2 Taxa of Schema Evolution for FOSS Projects [4]

For each of these projects, we created .tsv files, from the commit history GitHub provides. The .tsv files consist of 4 columns, Date - Author - NumAffectedFiles - Contains .sql. Date contains the date of a specific commit in GitHub, Author shows the username of the person that made the commit. The next column, NumAffectedFiles contains the number of all files committed. Finally, the Contains .sql column indicates how many of them were SQL files. It has to be noted that the

third column is a superset containing the fourth column. These files are used from ECE as input.

In our first attempt, we draw a bar for each commit of the .tsv file. This approach had two problems. First, for projects with a lot of commits, the exported bar chart was chaotic and second, it does not give the exact sense of how the project was maintained over time. We also wanted to monitor the abstention of commits and so on the absence of maintenance. To solve these problems, we decided to add a new feature to ECE. Using our existing .tsv files, the ECE can create new .tsv files that in each row have summed up all the commits for each month. In addition, for months with no commits, it adds zero lines. The new exported images are based on these new .tsv files. Next, to make it easier for the user to understand for each commit how many were code changes and how many of them were SQL changes, we plot above the x-axis only the number of source code changed, which means that these are no more a superset containing the number of SQL files changed. Above the x-axis, we plot only the number of SQL files changed.

Finally, to make it easier to check, compare and find patterns from the visual history of the projects, we decided to add to the ECE the ability to create a .html file for each taxon with all images.

3.1.3 Testing of EvolutionChartExporter

To use the EvolutionChartExporter tool and be sure that the exported bar charts are correct, we made two types of tests, first, we implemented two unit tests and second, we made visual tests for the exported images.

The unit tests we made were:

1. To check that the sum of commits for each month is correct. This JUnit test is implemented in the SumTest.java class in the test package of ECE.
2. To check that the months with zero commit have a zero value for the source code and SQL changes. This JUnit test is implemented in the AddZerosTest.java of the same package.

The way we implemented both these two tests is: First, we manually created files with all the possible extreme and bad cases we thought and believed could happen. After that, we manually created the files with the expected results from these tests

and the files we mentioned. In the end, we confirmed the equality from the expected files and the produced file results from the ECE to check the correctness of these Java classes.

The second type of test was to examine produced images visually. We made this type of test because the only way to check if the exported images correspond to the history from the .tsv files was by the eye. We randomly selected some of our projects and checked if the exported images correspond to the .tsv files. We also made this test to the exported images from our test .tsv files.

After all these tests we can say with confidence that ECE works properly and is safe to use for similar research.

3.2 Manual analysis of randomly selected projects from GitHub

In this subsection, we present our manual examination of six randomly selected projects. We selected some projects from different taxa for a deeper investigation to better understand the code and schema coevolution. We selected three projects from the Almost Frozen taxon, two from Focused Shot and Frozen taxon and one from moderate. The reason we choose these three taxa is that the more active is a taxon, there are more commits and schema changes to examine. For the same reason, we decided to examine three projects from the first taxon and only one from the third. As we will see, the last project had a huge difference in the commits required to examine compared to the projects from the first taxon. We expected and noticed that the number of commits to examining was increasing rapidly from one taxon to the other. We believe that to do that process multiple times for the next taxa is almost impossible.

Below, for each project we selected, we will place a table and make some cases and conclusions we came to, from our deep analysis. Each table has 6 columns. The first four columns are the same as the .tsv file. The next two columns show the state before and after the code and the schema changes.

For all the projects presented, we manually examined their commits. As the projects come with too many commits, we checked only the commits with a schema change and we decided to use a 'window' of ±3 commits from each schema change commit to observe the impact of those database changes on the source code. Also, we

examined all commits with the word fix or bug to see if it was a schema change that triggered the bug. This approach may lose some commits to source code related to the schema changes but decreased the time required to do the process dramatically. Next, in the tables, we show all the commits related to the schema changes and the commits made to the code to match these schema changes. We show also the commits triggered from a previous schema change to make fixies to the source code. We don't present here the commits related only to the code as we are interested in schema and source code coevolution.

Annotations:

When there are no changes from one commit to another, we will write *no change*. When this is written to the code, it means that in this commit, there were no changes to the code related to the database changes. So, the code related to the database is the same as the previous commit. For example, if a certain commit changed the way a function computes an algorithmic result, we are not interested in that. When we write *no change* to the DB, it means that in this commit the developers changed only the code (src part) related to the database (usually these commits are a bug fix). The '*no change*' refers always to the previous commit we present in the table and not the previous commit made to the GitHub project.

For each project, we tried to answer six core questions. These questions are:

1) What kind of changes happened to the schema and at the src to sync with schema evolution?
2) Why did these changes happen, for example, comments made to each commit?
3) When does schema evolution take place?
4) Where in the code/src is the impact of schema evolution and where is the maintenance effort located?
5) How do people change the schema and maintain the source code?
6) Who is related to the DB/src changes?

3.2.1 In-depth study of ALMOS_FROZEN projects

First, we selected three projects randomly from the ALMOST_FROZEN taxon.

These projects are:

1) joomlatools/joomla-platform-categories
2) umpirsky/tld-list
3) josephspurrier/gowebapp

1) joomlatools__joomla-platform-categories

About this project:

The project is a category extension for Joomlatools Platform (Joomlatools Platform is a modern Joomla stack that helps you get started with the best development tools and project structure). The description of the project is from GitHub. Joomla Categories is open-source software licensed under the GPLv3 license. The project uses PHP 7.0 and MySQL 5. The project started in 2015 and it was active for 3+ years, there are 63 commits made. The owner of the repository is the Joomlatools organization with 6 people and 47 repositories on GitHub.

Table 3.1 Commits related to the schema for the joomlatools/joomla-platform-categories project

Date YYYY-MM-DD	***Who***	***#Src*** updates	***#SQL*** updates	*State before*	*State after*
2015-07-08 03:42:35 +0200	Johan	55	2	***DB*** No DB	***DB*** Create 2 .sql files (create table/drop table #__categories). 1 table with 27 values.
				CODE No Code	***CODE*** 55 source code files. Raw Queries embedded.

2015-07-11 00:51:48 +0200	Johan	1	3	***DB*** No change	***DB*** - Rename install.mysql.sql to install.sql (file renaming). - Delete uninstall.mysql.sql file and create uninstall.sql. - Remove #__ prefix from database table names.
				CODE No change	***CODE*** Change references to the new file names.
2017-02-08 11:44:42 +0800	Allan	53	1	***DB*** No change	***DB*** Change sizes and encryption, e.g. - varchar(255) to varchar(400) - utf8 to utf8mb4 - utf8_bin to utf8mb4_bin
				CODE No change	***CODE*** - Change names, indentation, comments (2 md files not changed). - Refactor code. - New embedded SQL queries.

The removal of #__ prefix from table names (2015-07-11 commit) for the joomla-platform-categories cost no changes in the source code. This project is a part of the joomla-platform. After the manual search, we found that Joomla replaces the prefix, the commit was to match the 'parent' project. The source code is still using the prefix #__ for raw queries and uses the public static function getAssociations to get an array of associations between database tables and #__tableName.

1) ***What:*** Mostly code refactor.

2) ***Why:*** 1 was the initial commit and 2 commits with changes to match the joomla changes and joomlatools repository.

3) ***When:*** 2 commits at the beginning of the repository's life and 1 at the end.
4) ***Where:*** Commits related to com_category, it contains 7 packages and 3 files (commits were made to resource and controller packets and the 3 files).
5) ***How:*** Data type changes and renames.
6) ***Who:*** See the second column for more.
Johan: 2/3 (one was the initial commit)
Allan: 1/3

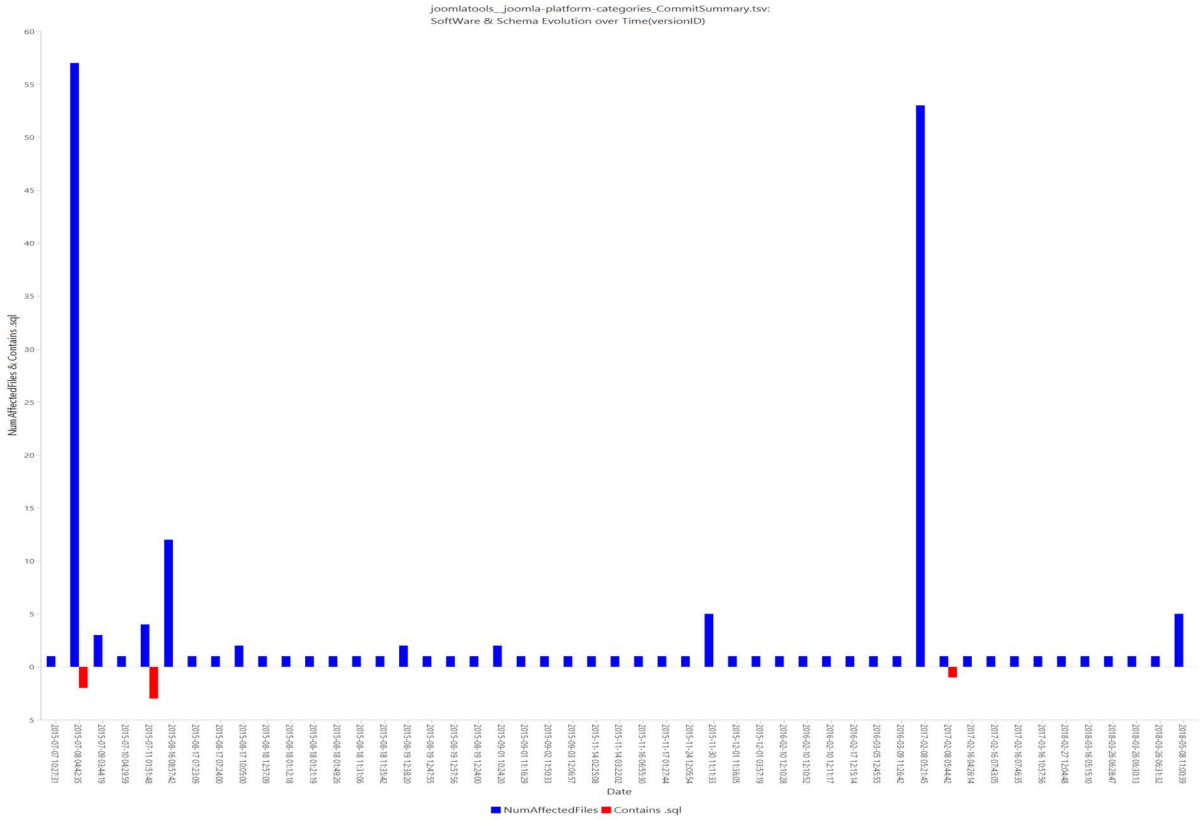


Figure 3.3 Schema and src commits for joomla-platform-categories, the image produced from the first version of ECE

2） umpirsky__tld-list

About this project:

This project is a huge list of all top-level domains (TLD) in all data formats. There is not much source code. The available formats are: Text - JSON - YAML - XML - HTML - CSV - SQL - MySQL - PostgreSQL - SQLite - PHP. The project started in 2016 and it was active for 2+ years, there are 12 commits made. The owner of the repository is Saša Stamenković with 226 repositories, 361 followers and 277 stars on GitHub.

Table 3.2 Commits related to the schema for the umpirsky/tld-list project

Date YYYY-MM-DD	***Who***	***# Src*** updates	***# SQL*** updates	*State before*	*State after*
2016-01-16 15:05:46 +0100	umpirsky	13	3	***DB*** No DB	***DB*** Creates 1 table(the same 3 times) in 3 SQL files (MySQL, PostgreSQL, SQLite) Inserts all tld domains
				CODE No Code	***CODE*** Same values in other formats (PHP, txt, JSON, HTML, etc)
2016-11-03 16:26:27 +0300	M	7	3	***DB*** No change	***DB*** Insert more values
				CODE No change	***CODE*** Insert the same values to no SQL format files
2018-02-27 19:56:16 +0100	Saša	8	3	***DB*** No change	***DB*** Insert more values (delete some)
				CODE No change	***CODE*** Insert/delete the same values to no SQL format files
2018-04-15 17:03:30 +0200	Saša	8	3	***DB*** No change	***DB*** Insert more values (delete some)
				CODE No change	***CODE*** Insert/delete the same values to no SQL format files

1) ***What:*** Insert new values, in different formats.

2) ***Why:*** To include more tld domains.

3) ***When:*** Commits made at the beginning-middle-end of the project life.

4) ***Where:*** Almost in all the files.

5) ***How:*** No schema changes.

6) ***Who:*** See the second column for more.

umpirsky: - 1/4 (one was the initial commit)

M: -1/4

Saša: - 2/4

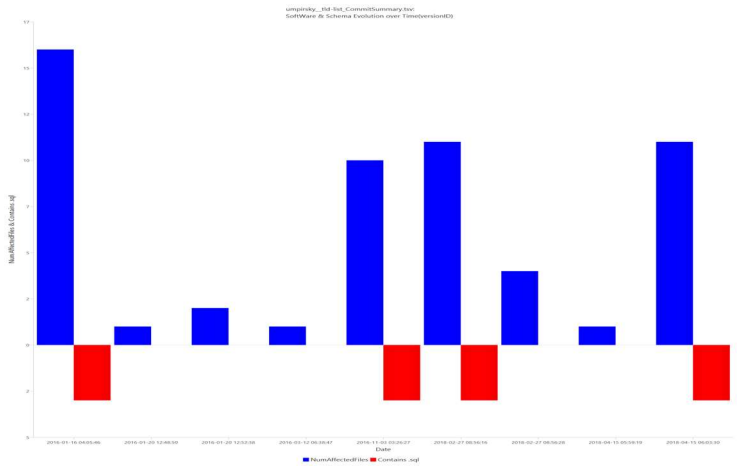


Figure 3.4 Schema and src commits for tld-list, the image produced from the first version of ECE

3) josephspurrier__gowebapp

About this project:

This project is a basic MVC (Model-view-controller) Web Application in Go. The web app has a public home page, authenticated home page, login page, register page, about page, and a simple notepad to demonstrate the CRUD operations, the description of the project is from GitHub (screenshots included on GitHub). The project started in 2015 and it was active for 2 years, there are 71 commits made. The owner of the repository is Joseph Spurrier with 50 repositories, 153 followers and 657 stars on GitHub.

Table 3.3 Commits related to the schema for the josephspurrier/gowebapp project

Date YYYY-MM-DD	***Who***	***# Src***	***# SQL***	***State before***	***State after***

		upd ates	upda tes		
2015-06-28 20:57:10 -0400	Joseph	34	1	***DB*** No DB	***DB*** First commit CREATE TABLE user_status CREATE TABLE user
				CODE No Code	***CODE*** First commit Add code
2015-07-04 02:00:09 -0400	Joseph	18	1	***DB*** No change	***DB*** Change path without file changes. database/database.sql → config/database.sql
				CODE No change	***CODE*** Changes to DB connection
2015-07-16 16:34:09 -0400	Joseph	0	1	***DB*** No change	***DB*** Update column sizes INT(10) to TINYINT(1) INT(1) to TINYINT(1)
				CODE No change	***CODE*** Update sizes in the code also. In Go language from int to uint32 or uint8.
2015-07-26 16:58:36 -0400	Joseph	6	2	***DB*** No change	***DB*** Change the default database to use SQLite (from MySQL to SQLite) Rename database (webframework => gowebapp) Add SQLite configuration file

				CODE No change	***CODE*** Add SQLite driver Change models/structures/SQLite case
2016-01-31 21:33:31 -0500	Joseph	4	1	***DB*** No change	***DB*** Remove SQLite, set MySQL again as the main DBMS
				CODE No change	***CODE*** Remove SQLite config file Remove SQLite case Add Bolt/Mongo DBs (as embedded GO files)
2016-04-24 11:09:13 -0400	Joseph	15	0	***DB*** No change	***DB*** No DB changes
				CODE No change	***CODE*** Updated variables names according to Lint (even db names/models)
2016-04-26 01:55:51 -0400	Joseph	1	0	***DB*** No change	***DB*** No DB changes
				CODE return u.ObjectId.Hex()	***CODE*** Fixing bug return u.ObjectID.Hex()
2016-04-26 02:59:18 -0400	Joseph	16	1	***DB*** No change	***DB*** CREATE TABLE note (6 var)
				CODE No change	***CODE*** Add note controller (CRUD) Delete unused models (one model/user.go for all three DBs)

2017-05-15 22:21:09 -0700	Shane	1	1	***DB*** SET storage_engine = InnoDB;	***DB*** SET default_storage_engine = InnoDB; to allow latest MySQL to work
				CODE No change	***CODE*** Change the absolute file path to relative

1) ***What:*** Code refactoring, change default dbms (x2 times), fix typos (e.g. 2016-04-26 commit), create a new table.

2) ***Why:*** Refactoring code and adding more information into the new table.

3) ***When:*** Uniformly DB commits into project's life.

4) ***Where:*** Usually commit changes to SQL files and all source code files using/related to it. There were also bug fixes into 2 files. When there were commits into the 2 SQL files, there were also commits to 4 specific src files. Sometimes, there were massive changes to files into model, controller, route and shared packages (e.g. when a new table was created).

5) ***How:*** Change default DBMS and add 1 table.

6) ***Who:*** See the second column for more.

Joseph: 8/9 (one was the initial commit)

Shane: 1/9

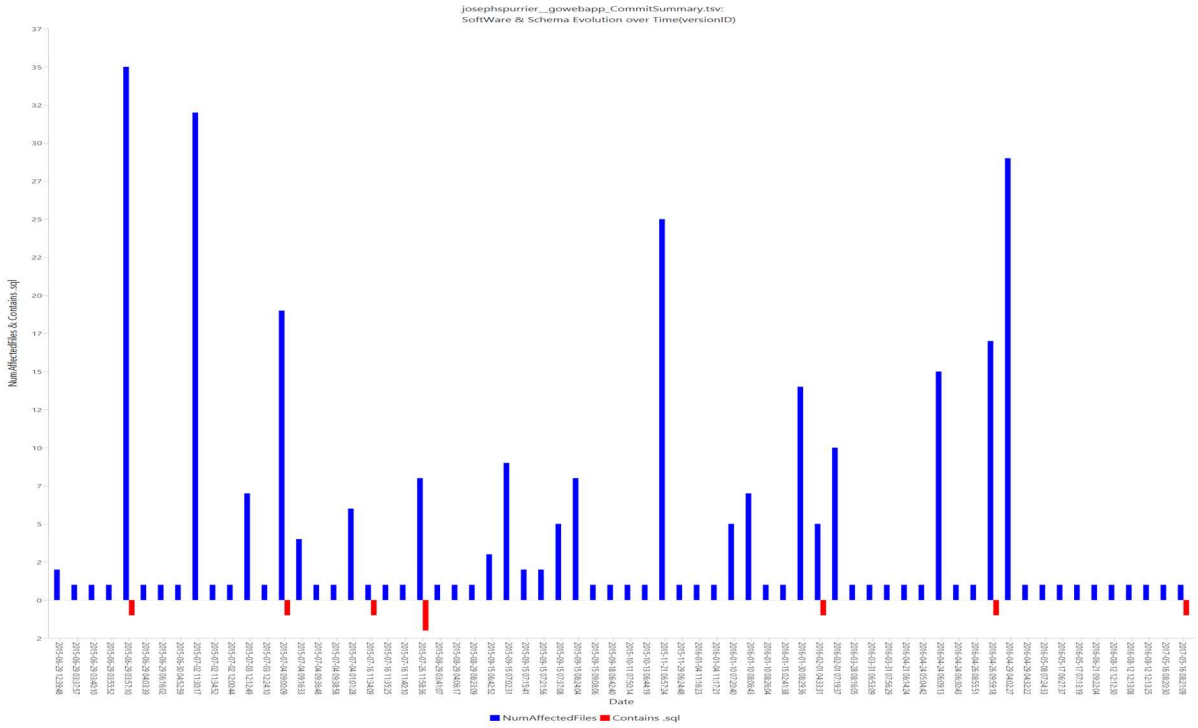


Figure 3.5 Schema and src commits for gowebapp, the image produced from the first version of ECE

3.2.2 In-depth study of FOCUSED-SHOT_n_FROZEN projects

Secondly, we selected three projects randomly from the FOCUSED-SHOT_n_FROZEN taxon.

These projects are:

1) accgit/acl

2) jasongrimes/silex-simpleuser

4) accgit__acl

About this project:

The project is a simple management of users' permissions. The project is written in JavaScript, PHP, Latte and CSS. The project started in 2017 and it was active for 2

years, there are 271 commits made. The owner of the repository is Zdeněk Papučík with 5 repositories, 5 followers and 27 stars on GitHub.

Table 3.4 Commits related to the schema for the accgit/acl project

Date YYYY-MM-DD	***Who***	***# Src*** updates	***# SQL*** updates	*State before*	*State after*
2017-05-23 13:08:53 +0200	Zdeněk	28	1	***DB*** No DB	***DB*** First commit CREATE TABLE privileges (2 var) CREATE TABLE resources (2 var) CREATE TABLE roles (3 var) CREATE TABLE permissions(5 var) Foreign keys to first 3 tables CREATE TABLE users (4 var) CREATE TABLE access (3 var) Foreign keys to users, roles --- Also Insert in all tables default values. DROP TABLE IF EXISTS for all 6 tables (in case of an update)
				CODE No Code	***CODE*** First commit Add code basically in PHP Include raw queries into php files for each table (ORM).

2017-06-02 07:12:44 +0200	Zdeněk	2	1	***DB*** INSERT INTO `access` (`id`, `role`, `user`) VALUES (NULL, ***3***, 1);	***DB*** INSERT INTO `access` (`id`, `role`, `user`) VALUES (NULL, ***2***, 1);
				CODE No change	***CODE*** Rename some classes and files. Changes not related to inserted values. Interesting fact that the cache.access was renamed to cache.acl -> see commit 2017-07-31 access table is renamed to acl.
2017-06-29 12:20:48 +0200	Zdeněk	0	1	***DB*** No change	***DB*** Remove all DROP TABLE IF EXISTS
				CODE No change	***CODE*** No change
2017-07-27 10:47:59 +0200	Zdeněk	0	1	***DB*** INSERT INTO `resources` (`id`, `name`) VALUES (NULL, 'Web:Web'), (NULL, 'Web:Login'), (NULL, 'Admin:Admin');	***DB*** INSERT INTO `resources` (`id`, `name`) VALUES (NULL, 'Web:Web'), (NULL, 'Web:Login'); Removes one default value
				CODE No change	***CODE*** No change
2017-07-27 13:11:00 +0200	Zdeněk	1	1	***DB*** No change	***DB*** INSERT INTO `resources` (`id`, `name`) VALUES (NULL, 'Web:Web'), (NULL, 'Web:Login'),

				(NULL, 'Admin:Admin'); INSERT INTO `roles` (`id`, `name`, `parent`) VALUES (NULL, 'guest', 0), (NULL, 'member', 1), (NULL, 'admin', 1); Inserted one default value in each table	
				CODE // Admin role that can do everything. $acl->addRole(self::ROLE_ADMIN); $acl->allow(self::ROLE_ADMIN, Security\Permission::ALL, Security\Permission::ALL);	***CODE*** // Admin role that can do everything. deletes one line in file acl/Authorizator.php $acl->allow(self::ROLE_ADMIN, Security\Permission::ALL, Security\Permission::ALL);
2017-07-27 13:12:00 +0200	Zdeněk	0	1	***DB*** INSERT INTO `roles`(`id`, `name`, `parent`) VALUES (NULL, 'guest', 0), (NULL, 'member', 1), (NULL, 'admin', 1);	***DB*** INSERT INTO `roles` (`id`, `name`, `parent`) VALUES (NULL, 'guest', 0), (NULL, 'member', 1), (NULL, 'admin', 2); Last was 1
				CODE No change	***CODE*** No change
2017-07-27 13:13:02 +0200	Zdeněk	0	1	***DB*** INSERT INTO `access` (`id`,	***DB***

				`role`, `user`) VALUES (NULL, ***2***, 1);	INSERT INTO `access` (`id`, `role`, `user`) VALUES (NULL, ***3***, 1); (view 2017-06-02 commit)
				CODE Xlo change	***CODE*** Xlo change
2017-07-31 11:31:57 +0200	Zdeněk	17	0	***DB*** No change	***DB*** No change
				CODE No change	***CODE*** Rename all 'id' to 'xxxxId' for the next commit (xxxx refers to table's name).
2017-07-31 12:09:40 +0200	Zdeněk	0	1	***DB*** ***Delete TABLE*** access (3 var) ***Renames*** CREATE TABLE `privileges` (`id` int(11)... CREATE TABLE `resources` (`id` int(11)... CREATE TABLE `roles` (`id`int(11)... CREATE TABLE `users` (`id`int(11)... CREATE TABLE `users` (`id` int(11)...	***DB*** ***CREATE TABLE*** acl (3 var) Actually, ***rename access to acl*** Move permissions TABLE on top of the file. Change id in INSERT for default values from NULL to number 1,2.. ***Renames*** 'id' -> '****Id' CREATE TABLE `privileges` (`privilegeId` unsigned... CREATE TABLE `resources` (`resourceId` unsigned... CREATE TABLE `roles` (`roleId` unsigned ...

					CREATE TABLE `users` (`userId` int(11) unsigned ...
				CODE No change	***CODE*** No change. Variable names in sync from the previous commit.
2017-08-01 07:04:03 +0200	Zdeněk	0	1	***DB*** No change	***DB*** Rearrange CREATE TABLE 'permissions' and 'acl' Add them both in the end of the file
				CODE No change	***CODE*** No change
2017-08-01 07:04:03 +0200	Zdeněk	0	1	***DB*** INSERT INTO `privileges` (`privilegeId`, `name`) VALUES (1, 'default');	***DB*** Fix indentation (add tabs) Make id in INSERT null from number, e.g., INSERT INTO `privileges` (`privilegeId`, `name`) VALUES (NULL, 'default');
				CODE No change	***CODE*** No change
2017-08-03 08:31:52 +0200	Zdeněk	0	1	***DB*** No change	***DB*** Commit just to add 1 tab
				CODE No change	***CODE*** No change
2017-08-03	Zdeně	0	1	***DB***	***DB***

08:33:40 +0200	k			(NULL, 'Admin:Admin'), (NULL, 'Web:Login'), (NULL, 'Web:Web');	INSERT INTO `resources` (`resourceId`, `name`) VALUES (NULL, 'Web:Web'), (NULL, 'Web:Login'), (NULL, 'Admin:Admin'); Rearrange...
				CODE No change	***CODE*** No change
2017-09-19 07:14:50 +0200	Zdeněk	1	1	***DB*** INSERT INTO `privileges` (`privilegeId`, `name`) VALUES (NULL, 'default');	***DB*** INSERT INTO `privileges` (`privilegeId`, `name`) VALUES ***(NULL, ':all')***, (NULL, 'default'); :all was not a default value
				CODE $row->privilege === ':all' ? $row->privilege = Security\Permission::ALL : $row->privilege;	***CODE*** const PRIVILEGE_ALL = ':all'; $row->privilege === self::PRIVILEGE_ALL ? $row->privilege = Security\Permission::ALL : $row->privilege;
2017-09-20 06:50:19 +0200	Zdeněk	0	1	***DB*** int(11) or int(10)	***DB*** Change data type for all tables to smallint(5)
				CODE No change	***CODE*** No change
2018-01-18 12:02:08 +0100	Zdeněk	0	1	***DB*** (NULL, ':all')	***DB*** (NULL, '.*') into table privileges

				CODE No change	***CODE*** No change
2018-01-18 12:07:04 +0100	Zdeněk	0	1	***DB*** (NULL, '.*')	***DB*** (NULL, ':all') into table privileges
				CODE No change	***CODE*** No change
2018-07-27 14:50:20 +0200	Zdeněk	0	1	***DB*** No change	***DB*** No change, make all inserts into one line
				CODE No change	***CODE*** No change

1) ***What:*** There are 6 tables created and ORM access to them. The developer removed drop tables if they exist. Most of the commits were: insert/remove default values or change them. Rename table/col_names (id->****Id) and classes/files. Change data types. Fix the indentation or rearrange the code lines.
2) ***Why:*** Most commits to change default values.
3) ***When:*** Most of them are at the beginning of the repository's life, but there are also commits at the middle and the end of the project's life.
4) ***Where:*** There is only one db.sql file. Usually, Object related (ORM) files with each table, or files/methods using them.
5) ***How:*** Usually to change default values, sometimes bug fixes, e.g. from renames.
6) ***Who:*** See the second column for more.
Zdeněk: 18/18 (one was the initial commit)

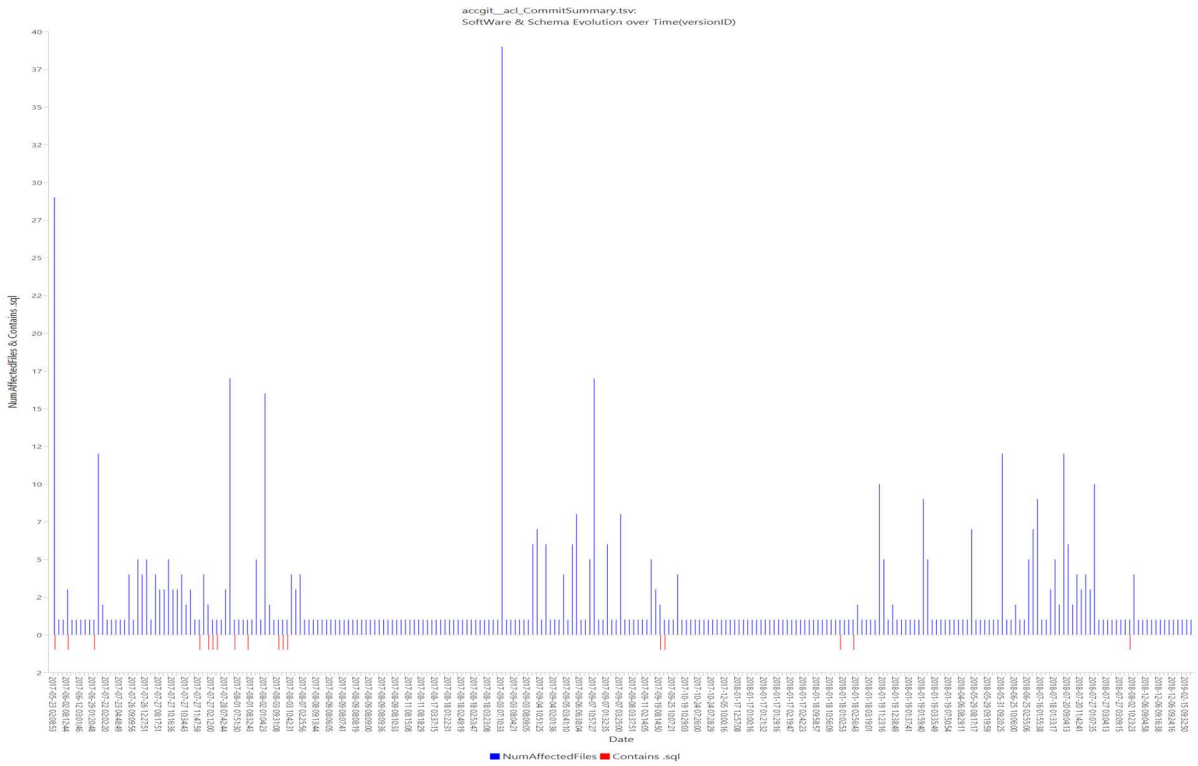


Figure 3.6 Schema and src commits for acl, the image produced from the first version of ECE

5) jasongrimes__silex-simpleuser

About this project:

A simple, extensible, database-backed user provider for the Silex security service. SimpleUser is an easy way to set up user accounts (authentication, authorization, and user administration) in the Silex PHP micro-framework. The project provides drop-in services for Silex that implement the missing user management pieces for the Security component. The project includes a basic User model, a database-backed user manager, controllers and views for user administration, and various supporting features. The description of the project is from GitHub. The project was written in PHP. The project started in 2013 and it was active for 3 years, there are 153 commits made. The owner of the repository is Jason Grimes with 35 repositories, 43 followers and 16 stars on GitHub.

Table 3.5 Commits related to the schema for the jasongrimes/silex-simpleuser project

Date YYYY-MM-DD	***Who***	***# Src*** updates	***# SQL*** updates	*State before*	*State after*
2013-04-14 14:53:22 +0000	Jason	6	1	***DB*** No DB	***DB*** CREATE TABLE users (7 var)
				CODE No Code	***CODE*** First commit
2014-08-24 08:56:21 -0400	Jason	2	1	***DB*** No change	***DB*** CREATE TABLE user_custom_fields (3 var)
				CODE No change	***CODE*** Add functions into the code to handle the new table
2014-08-24 09:18:33 -0400	Jason	0	1	***DB*** user_id INT(11) UNSIGNED NOT NULL ***AUTO_INCREMENT*** ... value VARCHAR(255) NOT NULL ***DEFAULT ”***	***DB*** Changes in the user_custom_fields table user_id INT(11) UNSIGNED NOT NULL ... value VARCHAR(255) ***DEFAULT NULL***
				CODE No change	***CODE*** No change
2014-08-24 09:31:08	Jason	0	1	***DB*** No change	***DB*** Add a new empty line between two tables in the sql file.
				CODE No change	***CODE*** No change

2014-08-24 10:02:09 -0400	Jason	0	1	***DB*** `password` VARCHAR(255) NOT NULL DEFAULT ”	***DB*** Changes in the users table `password` VARCHAR(255) DEFAULT NULL
				CODE No change	***CODE*** No change
2014-08-24 10:35:56 -0400	Jason	1	0	***DB*** No change	***DB*** No change

				CODE
				No change
					CODE
					Extra code for the new table when -> Reconstitute a User object from stored data
					if(!empty($data[’customFields’])){ $user->setCustomFields($data[’customFields’]);
					}

2014-09-04 00:52:31 -0400	Jason	5	1	***DB*** No change	***DB*** Add SQLite (same tables)
				CODE No change	***CODE*** Add DB tests for SQLite tables
2014-10-01 17:14:25 -0400	Jason	3	0	***DB*** No change	***DB*** No change
				CODE public function getUsername(){ return $this->email;	***CODE*** Add an optional username field, and allow logging in with either email or username. (Username is stored as a custom field for backward compatibility.)

					Return username, if not empty, otherwise the email public function getUsername(){ return $this->getCustomField('username') ?: $this->email;} See 2014-10-20 Jason commit (username was added the to db)
2014-10-20 01:52:10 +0200	enyosolutions	3	2	***DB*** No change	***DB*** Add for both MySQL & SQLite new columns. ***alter table*** users add username varchar(100) DEFAULT NULL;
				CODE No change	***CODE*** Add username in the code and change the structure of some functions.
2014-10-20 01:52:10 +0200	enyosolutions	2	2	***DB*** No change	***DB*** Re-commit the same changes with the previous commit into the 2 files.
				CODE No change	***CODE*** Re-commit the same changes with the previous commit into the 2 files.
2014-10-20 01:58:47 +0200	enyosolutions	1	0	***DB*** No change	***DB*** No change
				CODE No change	***CODE*** Re-commit the same changes from the.

					Next commit cancels this commit.
2014-10-20 23:17:27 -0400	Jason	3	2	***DB*** No change	***DB*** Add 4 columns into the users table, for both mySql and sqLite.
				CODE No change	***CODE*** Add new fields to src objects related to table mapping (ORM).
2014-10-21 00:44:53 +0200	enyosolutions	2	2	***DB*** No change	***DB*** Cancel the changes (alter...) made in the previous commit.
				CODE No change	***CODE*** Cancel the changes into the 2 from the 3 files changed in the previous commit. Cancels were made by Jason (conflicts) who started the project.
2014-10-25 16:19:26 -0400	Jason	3	1	***DB*** No change	***DB*** Add support for migrating the database from version 1.x to 2.0 and back again. Add-v1 SQLite
				CODE No change	***CODE*** Add readme and code (and test) to help migration from V1 to V2.
Few update in .md				***DB*** No change	***DB*** No change
				CODE No change	***CODE*** Few updates, not related to schema

2014-10-28 06:38:16 -0400	Jason	1	0	***DB*** No change	***DB*** No change
				CODE 'username' => $user->getUsername()	***CODE*** ***Fix bug*** causing email address to be stored as username 'username' => $user->getRealUsername() View 2014-10-20 Jason commit.

1) ***What:*** Changes happened: create/add table, data type changes, add new DBMS, add columns, add support for migration. Also, after a schema change, commits were made to src, at the same commit or at the same day. There was one bug fix 8 days after the db changed (last commit).

2) ***Why:*** To update schema with more info, add DBMS and migration ability.

3) ***When:*** At the beginning, middle and at the end of the project's life.

4) ***Where:*** Usually to the same files (2 sql files, src files related to user model and test files for the db).

5) ***How:*** Make changes in the db schema, add dbms and migration support.

6) ***Who:*** See the second column for more.

Jason: 11/15 (one was the initial commit)

enyosolutions: 4/15 (commits canceled due to conflicts)

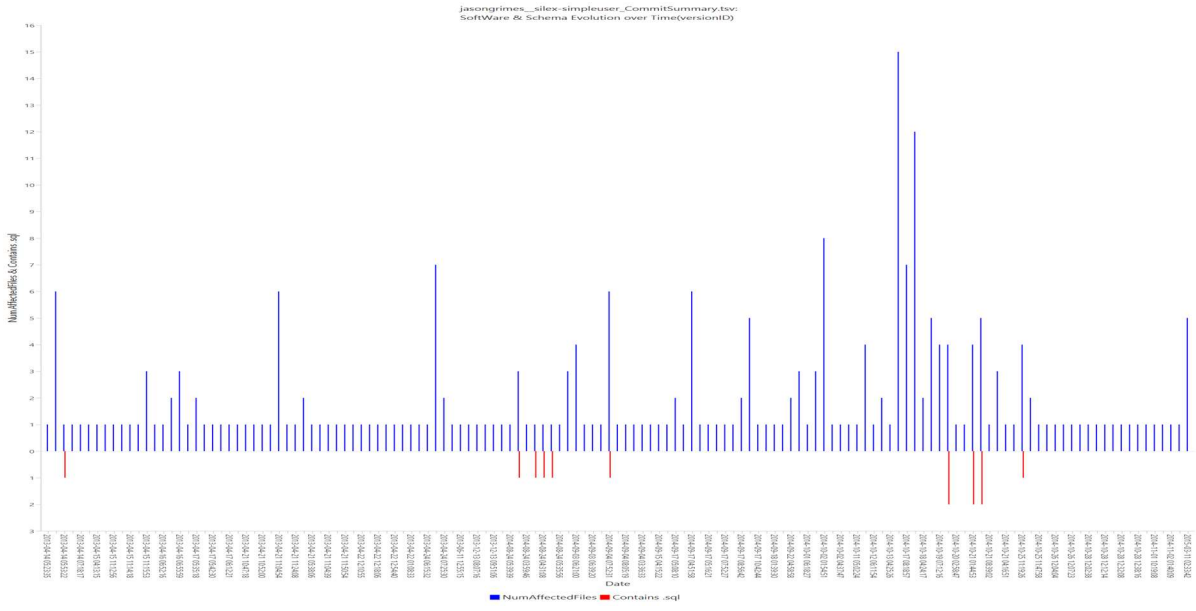


Figure 3.7 Schema and src commits for silex-simpleuser, image produced from the first version of ECE

3.2.3 In-depth study of MODERATE project

Finally, we selected one project randomly from the MODERATE taxon.

This project is:

1) mapbox/osm-comments-parser

6) mapbox__osm-comments-parser

About this project:

The project consists of parsers to read Notes and Changeset XML files and save them in a Postgres DB. The project was written in JavaScript. The project started in 2015 and it was active for 2 years, there are 183 commits made. The owner of the repository is the Mapbox organization with 55 people and 849 repositories on GitHub.

Table 3.6 Commits related to the schema for the mapbox/osm-comments-parser project

Date YYYY-MM-DD	***Who***	***# Src*** updates	***# SQL*** updates	*State before*	*State after*
2015-11-10 11:51:44 +0530	Sanjay	18	1	***DB*** No DB	***DB*** First commit CREATE TABLE IF NOT EXISTS users (var 2) notes (var 5) note_comments (var 6) changesets (var 7) changeset_tags (var 7) changeset_comments (var 5)
				CODE	***CODE***

				No Code	First commit, insert readme and source code, mostly in javascript(1 js file for each db table to handle, ORM). Test files added also.
2015-11-13 19:36:07 +0530	Sanjay	2	0	***DB*** No change	***DB*** No change
				CODE No change	***CODE*** Fix, add opened_by user as an attribute of notes Add ***var _ = require('underscore');*** and function to handle it, change functions that where using node table
2015-11-16 13:03:58 +0530	Sanjay	3	0	***DB*** No change	***DB*** No change
				CODE var userID = comment.UID ‖null; var userName = comment.USER ‖ null; var timestamp = comment.DATE;	***CODE*** Fix bug of not saving discussion users and timestamps correctly var userID = comment.attributes.UID ‖ null; var userName = comment.attributes.USER ‖ null; var timestamp = comment.attributes.DATE;
2015-11-19 12:05:42 +0530	Sanjay	3	2	***DB*** Into create_tables.sql	***DB*** UPDATE changesets SET bbox = ST_MakeEnvelope(min_lon, min_lat,

				1 - Table changesets user_id integer REFERENCES users (id), bbox geometry(POLYGON, 4326) ***2 - Table changeset_tags*** changeset_id integer REFERENCES changesets (id) ***3 - Table changeset_comments*** changeset_id integer REFERENCES changesets (id), user_id integer REFERENCES users (id) NULL	max_lon, max_lat, 4326); ***1 - Table*** user_id integer, min_lon float NULL, min_lat float NULL, max_lon float NULL, max_lat float NULL, bbox geometry(POLYGON, 4326)NULL ***2 - Table*** changeset_id integer ***3 - Table*** changeset_id integer, user_id integer NULL Generate CSV files that can be \Copied into postgres, refs ADD file changesets/post_initial.sql
				CODE Xlo change	***CODE*** Add changesets/csv.js and change functions for the db changes
2015-11-23 14:14:24 +0530	Sanjay	0	1	***DB*** No change	***DB*** Add SQL file to create indexes CREATE INDEX changesets_created_at_idx ON changesets(created_at); CREATE INDEX changesets_closed_at_idx ON

					changesets(closed_at) ; +++ more
				CODE No change	***CODE*** No change
2015-12-09 16:47:53 +0530	Sanjay	0	2	***DB*** No change	***DB*** move sql files to scripts/ folder create_indexes.sql → scripts/create_indexes.sql create_tables.sql → scripts/create_tables.sql
				CODE No change	***CODE*** Xlo change
2015-12-17 14:55:18 +0530	Sanjay	4	1	***DB*** No change	***DB*** Table changesets Add: discussion_count integer
				CODE No change	***CODE*** Change functions. Populate and write test. Change indentation.
2015-12-18 11:40:43 +0530	Sanjay	0	1	***DB*** No change	***DB*** Create index on discussion_count and comments timestamp (for sorting) CREATE INDEX changesets_discussion_count_idx ON changesets(discussion_count); CREATE INDEX changeset_comments_timestamp_idx ON

					changeset_comments(timestamp);
				CODE No change	***CODE*** No change
2016-01-20 15:00:10 +0530	Sanjay	1	0	***DB*** No change	***DB*** No change
				CODE var updateQuery = 'UPDATE notes SET created_at=$2, closed_at=$3, point=ST_GeomFr omText($4, 4326) where id=$1';	***CODE*** Update note if closed_at has changed var updateQuery = 'UPDATE notes SET created_at=$2, closed_at=$3, ***opened_by=$4***, point=ST_GeomFrom Text($5, 4326) where id=$1';
2016-02-23 13:31:37 +0530	Sanjay	0	1	***DB*** No change	***DB*** changesets/post_initia l.sql \COPY users(id,name) FROM 'csv/users.csv' DELIMITERS ',' CSV;
				CODE No change	***CODE*** No change
2016-11-29 17:15:15 +0530	Sanjay	5	2	***DB*** \COPY changesets(id, created_at, closed_at, is_open, user_id, num_changes, min_lon, min_lat, max_lon, max_lat) FROM 'csv/changesets.csv' DELIMITERS ',' CSV;	***DB*** Save is_unreplied boolean, add to schema \COPY changesets(id, created_at, closed_at, is_open, user_id, num_changes, ***is_unreplied***, min_lon, min_lat, max_lon, max_lat)

					FROM 'csv/changesets.csv' DELIMITERS ',' CSV; Table changesets add is_unreplied boolean
				CODE No change	***CODE*** Populate new value and fix tests.
2016-11-29 17:53:21 +0530	Sanjay	1	0	***DB*** No change	***DB*** No change

				CODE
				No change
					CODE
					Add utils
					module.exports = {};
					module.exports.getIsUnreplied = getIsUnreplied;
					function ***getIsUnreplied***(uid, comments) {
					var lastComment = comments.slice(-1)[0];
					if (lastComment.attributes.UID === uid) {
					return false;
					} else {
					return true;
					}
					}

2016-11-29 18:33:58 +0530	Sanjay	2	2	***DB*** No change	***DB*** Add to changesets username text Add it to \COPY changesets(...
				CODE No change	***CODE*** Populate new field
2016-11-30 11:38:29 +0530	Sanjay	4	3	***DB*** No change	***DB*** Deletes ***changeset_tags*** table

					Add to ***changesets*** table comment text NULL, source text NULL, created_by text NULL, imagery_used text NULL, Delete \COPY changeset_tags(... Remove 4 indexes about changeset_tags_... Add CREATE INDEX changesets_comment_tsvector_idx
				CODE No change	***CODE*** Remove functions using the deleted table (***changeset_tags***) and change functions handling the changed table (***changesets***). Add getChangesetTags() to utils (retrieves the 4 new inserted values to ***changesets*** table)
2016-12-01 12:34:35 +0530	Sanjay	1	1	***DB*** No change	***DB*** Add to changeset_comments username TEXT NULL
				CODE No change	***CODE*** Add variable and function using the table (changesets/db.js)

2016-12-01 12:39:50 +0530	Sanjay	1	1	***DB*** No change	***DB*** Add \COPY changeset_comments(...
				CODE No change	***CODE*** Handle username in changeset_comments for initial csv generation. Add attribs.USER ? attribs.USER : null
2016-12-01 12:41:44 +0530	Sanjay	0	1	***DB*** No change	***DB*** Create indexes on username fields
				CODE No change	***CODE*** No change
2017-01-16 17:45:52 +0530	Sanjay	4	2	***DB*** No change	***DB*** Add columns to users table name text, first_edit timestamptz, changeset_count integer, num_changes integer Add to post_initial \COPY users(...
				CODE No change	***CODE*** Add additional user metadata also to functions
2017-01-25 12:17:17 +0530	Sanjay	6	2	***DB*** No change	***DB*** Add to \COPY changesets(.., discussion_count,... CREATE INDEX changesets_is_unreplied_idx ON changesets(***is_unreplied***);

					Field was added 2015-12-17
				CODE No change	***CODE*** Add it to csv file and make/fix tests
2017-02-01 15:21:58 +0530	Sajjad	3	1	***DB*** No change	***DB*** CREATE TABLE IF NOT EXISTS stats (var 10)
				CODE objects/objUser.js tags: {}	***CODE*** Change function countTags(users, obj) tags_modified: {}, tags_created: {}, tags_deleted: {}
2017-02-01 16:28:42 +0530	Sajjad	4	1	***DB*** id integer PRIMARY KEY	***DB*** Change into table stats id serial PRIMARY KEY
				CODE No change	***CODE*** Create objects/db.js to write/save changes to db, rename some variables
2017-02-01 16:54:06 +0530	Sajjad	5	1	***DB*** first_edit timestamptz	***DB*** Into table users (nullable) first_edit timestamptz ***NULL***
				CODE callback(userID)	***CODE*** Fix: callback(null, userID); Rename variables, ***delete*** 2 js files,use userModel instead of objUser.js and tags.js (not used any more,

					see next commit, no need to filter tags)
2017-02-01 18:04:08 +0530	Sajjad	6	0	***DB*** No change	***DB*** No change
				CODE No change	***CODE*** Creates changes/user-model.js Used in last commit (deleted changes/objUser.js) Basic tests
2017-02-03 15:09:31 +0530	Sajjad	1	1	***DB*** changesets integer NULL	***DB*** Table stats changesets integer ARRAY
				CODE val.changesets = _.size(_.uniq(val.changesets));	***CODE*** val.changesets = _.uniq(val.changesets);
2017-03-28 11:31:59 +0530	Kushan	1	0	***DB*** No change	***DB*** No change
				CODE var firstEditDate = new Date(userRow.first_edit) ? userRow.first_edit : null;	***CODE*** In users/db.js file var firstEditDate = userRow.first_edit ? new Date(userRow.first_edit) : null; After the 2017-02-01 commit.
2017-03-31 14:01:40 +0530	Kushan	2	0	***DB*** No change	***DB*** No change
				CODE var checkUserQuery = 'SELECT id, name,	***CODE*** Add ***first_edit*** to user select query, wasn't retrieved ->

				changeset_count, num_changes from users where id=$1’;	was always null (see previous commit) var checkUserQuery = ’SELECT id, name, changeset_count, num_changes, ***first_edit*** from users where id=$1’;
2017-04-07 16:55:35 +0530	Kushan	8	1	***DB*** id serial PRIMARY KEY	***DB*** Table stats id uuid PRIMARY KEY
				CODE No change	***CODE*** No src changes related to db changes. From the commit comment: Fix duplicate stats data (adds replicationId), add tests.
2017-04-07 17:55:17 +0530	Sanjay	0	1	***DB*** No change	***DB*** Add indexes CREATE INDEX stats_change_at_idx ON stats(change_at); CREATE INDEX stats_uid_idx ON stats(uid);
				CODE No change	***CODE*** No change
2017-08-03 16:38:23 +0530	Sajjad	3	1	***DB*** No change	***DB*** Table stats add rows nodes_created bigint ARRAY, ways_created bigint ARRAY, relations_created bigint ARRAY, nodes_modified bigint ARRAY,

					ways_modified bigint ARRAY, relations_modified bigint ARRAY, nodes_deleted bigint ARRAY, ways_deleted bigint ARRAY, relations_deleted bigint ARRAY
				CODE No change	***CODE*** Add new variables from table to to the code and to counter.js
2017-08-04 12:26:02 +0530	Sajjad	1	0	***DB*** No change	***DB*** No change
				CODE No change	***CODE*** Add new field from stats table to update query

1) ***What:*** Create/delete tables not only at the beginning as usually for the previous projects, also add/remove src code for these tables. Fix bugs into src occurred by schema changes. Add features to the project. Move SQL files to a folder. Add/delete attributes/columns into tables. Multiple data type changes in the project's life.

2) ***Why:*** Add features to the project, bug fixes and code refactor.

3) ***When:*** Uniformly spread commits.

4) ***Where:*** Usually the same files (group of files).

5) ***How:*** Schema changes and src maintenance mostly.

6) ***Who:*** See the second column for more.

Sanjay 27/30 (one was the initial commit)

Kushan 3/30

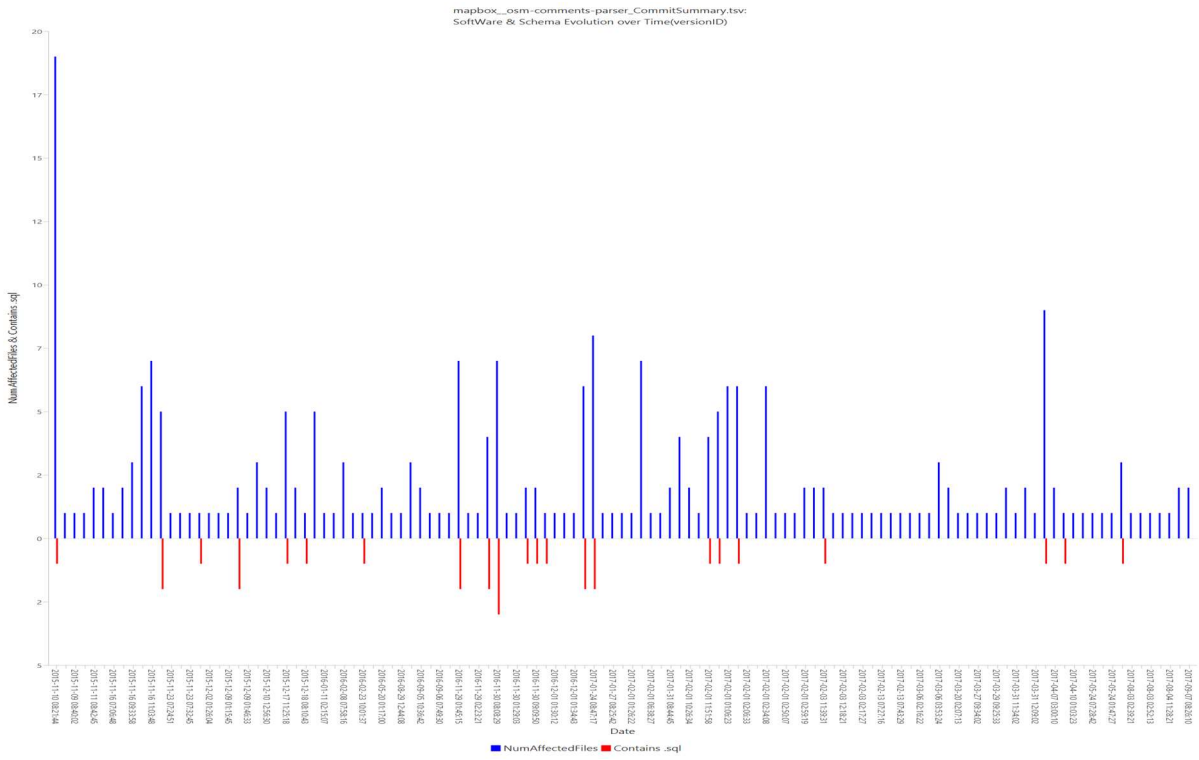


Figure 3.8 Schema and src commits for osm-comments-parser, the image produced from the first version of ECE

3.3 Results and findings from deep investigation

In this section, we group our findings and answer the six main questions mentioned before for all projects together. We also locate and export patterns of how schema and source code coevolve, for example frequently affected packages, how the schema life compares to the source code life.

In general, we observed that at the very first commits, the developers uploaded a large number of files and then mostly change and edit these files. That indicates that possibly before the use of GitHub, developers have been working into the project 'locally', so, we have lost bug fixes and possibly schema changes. This phenomenon seems to be more intense for the almost_frozen taxon, and the more active a taxon was, the more the project has been developed progressively.

1) ***WHAT***

We have mapped the schema and src changes for a better understanding of what each group of schema changes cost to source code. In Figure 3.9, we depict at the left column the projects we investigated. In the middle column, we depict the schema changes we found in these projects and in the right column we depict the changes that occurred to the source code. For each project, we used a different color to color the project box border and arrows to schema changes (e.g. red color for the joomla-platform-categories). We can summarize the schema changes and the source code changes we found into fifteen and nine types respectively. The schema change types are: File rename, File relocation, Update datatypes, Insert values (rows), Switch DBMS, Create a new table, Delete table, Change of the storage engine, Correcting/Updating previous values, Rename attributes, New DBMS added, Add attributes (columns), Delete attributes (columns), Index, No schema changes. The source code change types are: Changes unrelated to schema changes, Keep src in sync with the new values, Sync sizes in code, Sync code, Table controller added, cleanups' of table models/code, Add DB tests, Fix bugs triggered from schema changes (previous) and Various changes.

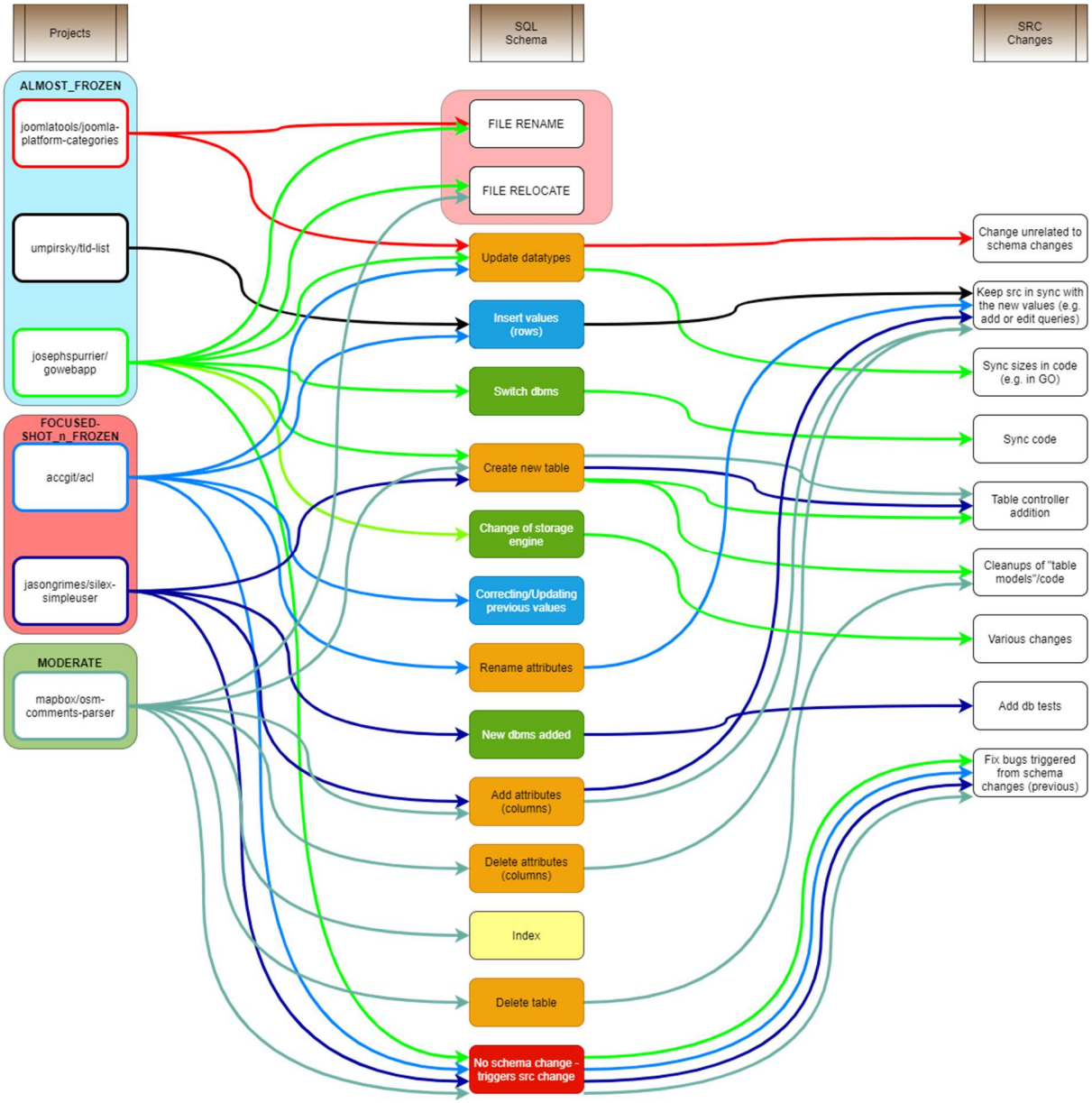


Figure 3.9 Analysis of schema changes per project and their impact on source code

In Figure 3.9, to make it easier to find patterns, we categorized the schema changes to a higher level. We can see where each schema change belongs from the colors the blocks are colored.

These categories are:

1) Schema change @logical level (orange).
2) Change @accompanying data in the Data Definition Language File (blue).
3) Change @engine supported (green).
4) Change @accompanying database code (red).
5) Change @physical level (yellow).

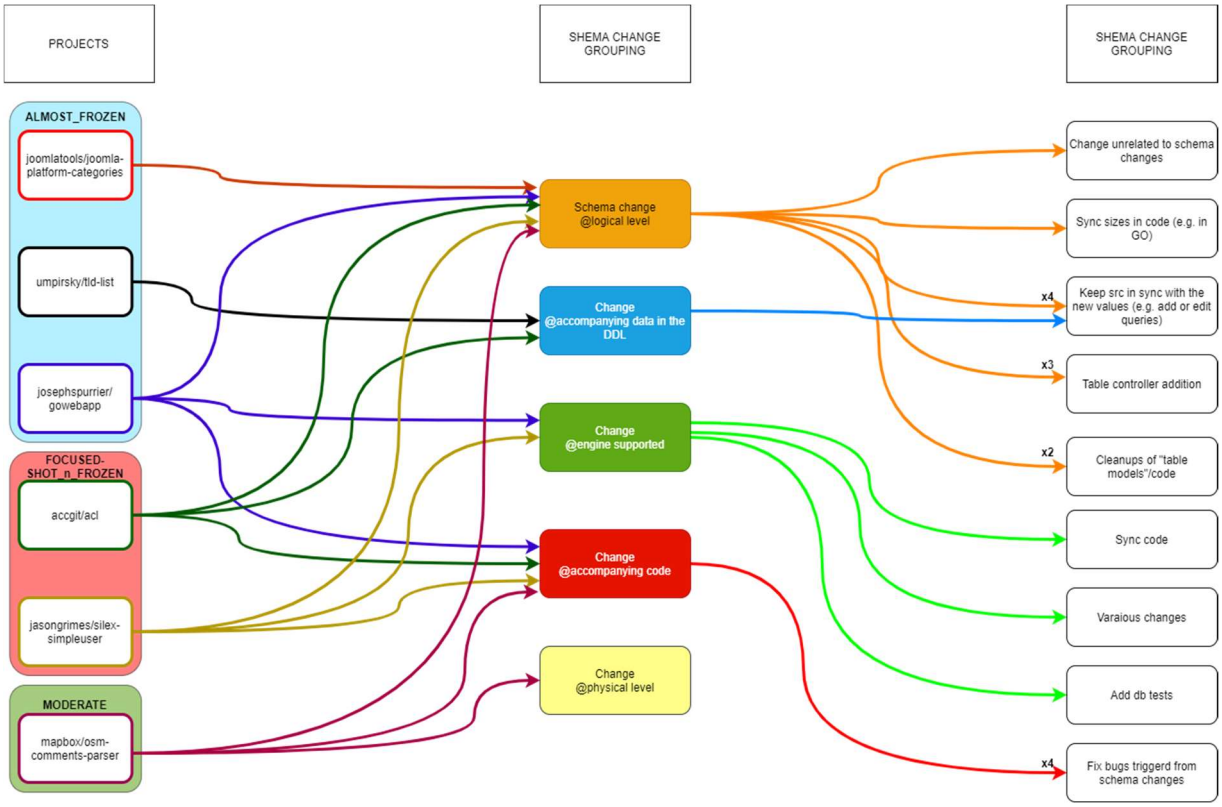


Figure 3.10 Grouped schema changes and their impact on source code

In Figure 3.10 we can clearly identify what impact has each group of schema change to the source code.

2) WHY

From our investigation of the six previous projects, we found out that in general, the most commits were to insert new values, e.g. new default values, to add more features to the project (e.g. a new table to save extra information or the user) and code refactoring or bug fixes after schema changed.

Table 3.7 Reasons schema changes happen to each project

Project	***Reasons***
1)joomlatools__joomla-platform-categories	Changes to match joomla changes and joomlatools repository.
2)umpirsky__tld-list	To include more tld domains (add rows).

3)josephspurrier__gowebapp	Refactor the code and add extra information (a new table note was added).
4)accgit__acl	Most commits were to change default values (inserted rows e.g. admin).
5)jasongrimes__silex-simpleuser	Update schema with more info, add new DBMS and migration ability.
6)mapbox__osm-comments-parser	Add new features, bug fixes and code refactoring.

3) WHEN

At the six projects we examined, we found that there were commits to the schema and the source code while the project was alive. On most of them, the commits were uniformly spread in relation to the project's life.

Table 3.8 When schema commits happened to each project

Project	***When schema commits happened***
1)joomlatools__joomla-platform-categories	Two commits at the beginning of the project's life and one at the end.
2)umpirsky__tld-list	Three commits, one at the beginning, one in the middle and one at the end of the project's life.
3)josephspurrier__gowebapp	Uniformly spread commits to the database in the project's life.
4)accgit__acl	Most of the commits were at the beginning of the project, but there are also commits at the middle and the end of the project's life.
5)jasongrimes__silex-simpleuser	Uniform commits, at the beginning, in the middle and at the end of the project's life.
6)mapbox__osm-comments-parser	Uniformly spread commits.

6) WHO

From what we can see, the projects with more active schema evolution, tend to have most of the commits made to the project concentrated to one person.

Table 3.9 Who made schema commits to each project

Projects	Percentage (%) of developers committing schema changes	Percentage (%) of commits made by the developer with the highest percentage of changes
joomlatools__joomla-platform-categories	50%	66.6%
umpirsky__tld-list	100%	50%
josephspurrier__gowebapp	66.6%	88.8%
accgit__acl	50%	100%
jasongrimes__silex-simpleuser	25%	73.3%
mapbox__osm-comments-parser	50%	90%

CHAPTER 4

CUMULATIVE ANALYSIS OF SCHEMA AND CODE CO-EVOLUTION

In the following chapter, we provide a cumulative analysis of our dataset in order to be able to answer the two research questions we introduced in chapter 1. We added to the EvolutionChartExporter the ability to compute and visualize the cumulative activity as we will show in the next sections. In the next sections of this chapter, we present how we have obtained the required results and the algorithms we used. Finally, we discuss our findings on the two research questions we made at the beginning of this thesis.

4.1 Cumulative analysis and algorithm

4.1.1 Introduction to cumulative analysis

For our study in schema and software co-evolution, we made a cumulative analysis of the activity for each project in every taxon. The cumulative percentage is a running total of the percentage values occurring across a set of responses. The total will either remain the same or increase, reaching the highest value of 100% after totalling all of the previous percentages. For example, if the percentage of a project's progression in 4 quarters of a year is 40%, 25%, 20%, and 15%, respectively, the cumulative percentage values would be 40%, 65%, 85%, and 100%, for each quarter.
The formula for the cumulative percentage is as follows:

$$cumPct_i = \frac{1}{TotalActivity} \sum_{k=0}^{i} activity_k = cumPct_{i-1} + \frac{activity_i}{TotalActivity}$$

with $acitvity_k$ being the activity in the k-th time unit, and *TotalActivity* is the total amount of activity measured for the entire lifetime of a project.
The above formula obviously applies to all kinds of activity measurements, like *projectActivity*, *schemaEvolutionActivity*, *Expansion*, *Maintenance*, etc.
Using the files from chapter 3, to find the projects duration and total activity and the exported MonthlySchemaStats files from the Heraclitus (on GitHub: *https://github.com/pvassil/HeraclitusFire*), we created a new file for each project with the computed cumulative activities. The new file consists of six columns, these are: Month, SchActivity, PrjActivity, cumulPtime, cumulSchActivity, cumulPrjActivity.

- The first column counts from 0…n, with n the project life in months.
- The SchActivity column, consists of the attributes changed from the commits made in the month i, this value is computed by Heraclitus and is located in the TotalAttrActivity column of the MonthlySchemaStats file.
- The PrjActivity counts the number of files that changed in the commits made the month i (contains source code and database files).
- The cumulPtime contains the percentage of the projects' life until the month i.

- The cumulSchActivity column contains the cumulative percentage of the SchActivity column over time.
- Finally, the cumulPrjActivity column contains the cumulative percentage of PrjActivity values over time.

In section 4.2 we will explain the tool we created to compute the cumulative analysis and the tests we made.

4.1.2 Algorithm of cumulative analysis

In the previous section, we gave a definition of what a cumulative percentage is. In this section, we present the algorithm that we used for our research and we implement it in the EvolutionChartExporter. The main feature of the cumulative percentage is the use of the previous value [i-1] to find the current value [i].

- ***totDur*** is the total duration/life of the project in months
- ***totPrjAct*** is the total number of changed files, the sum of changed files in every commit, contains the source code files and the database files (sum of all prjActivity[]).
- ***totSchAct*** is the total number of changed attributes, the sum of changed attributed according to the MonthlySchemaStats file exported from HeraclitusFire (sum of all SchActivity[]).
- ***prjActivity*[]** is an array with the number of the changed files every month.
- ***SchActivity*[]** is an array with the number of changed attributes each month.

The algorithm we implemented is introduced in Algorithm 4.1.

Algorithm Computation of the Cumulative percentage	
1:	int totDur = getTotalDuration()
2:	int totPrjAct = getTotalPrjActivity()
3:	int totSchAct = getTotalSchemaActivity()
4:	
5:	prjActivity[] = getPrjActivity()
6:	SchActivity[] = getSchActivity()

```
7:
8:  cumulPrjActivity[0] = prjActivity[0] / totPrjAct
9:  cumulSchActivity[0] = SchActivity[0] / totSchAct
10: cumulPTime[0] = 0
11:
12: for each month i in 1..totDur
13:     cumulPrjActivity[i] = cumulPrjActivity[i-1] + (PrjActivity[i]
    /TotPrjActivity)
14:     cumulSchActivity[i] = cumulSchActivity[i-1] +
    (SchActivity[i]/TotSchActivity)
15:     cumulPtime[i] = i / totDur
16: end for
```

Algorithm 4.1 Computation of cumulative percentage

From the definition of cumulative percentage, we can see that all cumulative variables cumulPrjActivity[n], cumulSchActivity[n], cumulPtime[n], with n equals to the last month, have to be 1.0 (or 100%).

4.1.3 A comment on the generation of Monthly Schema Stats

Before we continue and present the tool we made to compute and visualize the cumulative percentage, we will open a parenthesis to make a comment on how Heraclitus produce the schema monthly stats and why we should know it. Heraclitus produces two kinds of statistics:

- Evolutionary statistics for the heartbeat of the schema evolution, in which, the originating version of the schema life is not included: the aim of these statistics is to quantify how much the schema has changed after its birth.
- Monthly statistics for the heartbeat of the schema evolution, that compute the number of changes for each month, and, in which, the originating version is included.

Thus, the total sum of activity changes is different in these two kinds of evidence, and differs with respect to the number of attributes born in the originating version. In the rest of our deliberations, we will refer to the monthly stats, as this is the

respective measure that we can use to compare against the monthly stats of the project activity.

4.2 Expanding of EvolutionChartExporter

4.2.1 How EvolutionChartExporter computes and visualize the cumulative activity of the projects

In this chapter, we will present the extension we incorporated into the first version of EvolutionChartExporter, as presented in section 3. The structure of the EvolutionChartExporter remained the same, although we added the ability to the software to create line charts this time.
In more detail, we implemented the next new classes:

- ***ComputeCumulativeEngine:*** This is an engine class, responsible to compute the cumulative percentage and extorting it to a file.
- ***CumulativeDataLoader:*** This class was created to load the required files for each project (CommitSummary and MonthlySchemaStats files). These files are important to find the activities of each project and schema, as we saw in the algorithm before. The loader uses the AddZeroEngine implemented in section 3, to add zero months with no activity, so the schema activity can match the project activity.
- ***CumulativeModel:*** This class is an object that contains the six values we save to our cumulative file (these values are: Month, SchActivity, PrjActivity, cumulPtime, cumulSchActivity, cumulPrjActivity). Each object is an instance of an activity month (one line in the file).
- ***LineChartExporter:*** With this class, we were able to export the line chart images. The line chart contains two lines, one for the project activity and one for the schema activity.
- ***ProduceCumulativeImageEngine:*** It is using the LineChartExporter to create the images. For each project, we produce two images, one with a percentage of the time, using the cumulTime column, and the other with absolute time, using the month column.

Figure 4.1 shows a flow chart of how EvolutionChartExporter creates the cumulative files and images.

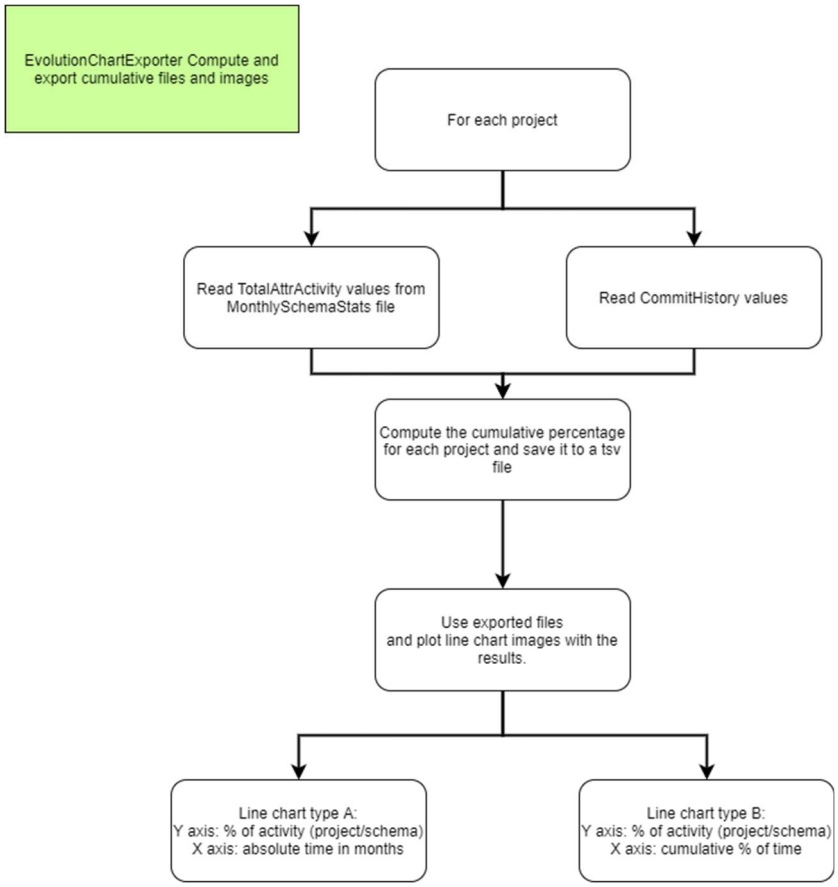


Figure 4.1 EvolutionChartExporter flow chart.

Figure 4.2 shows the class diagram of the EvolutionChartExporter. Only the functionality classes are shown here, the JUnit class tests are not shown.

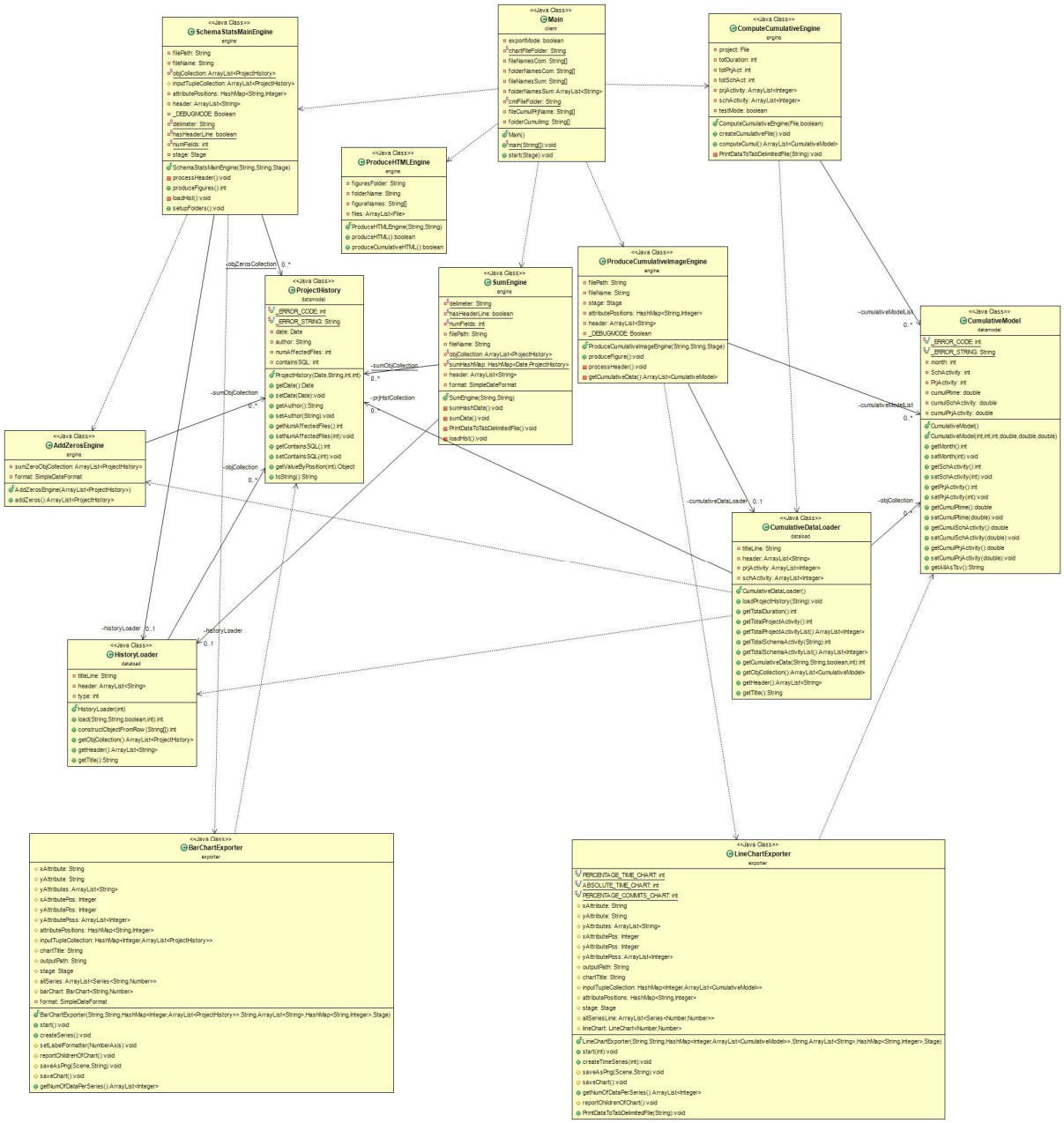


Figure 4.2 Class diagram of EvolutionChartExporter.

In Figure 4.3, we can see an example of a line chart image exported from the tool we created. In the image, we can see that both of the lines are always increasing (or remaining the same, never decreasing). Also, both lines end up at 100%, in the Y-axis is 1.0 (represents the 100%). With the blue dotted line, we depict the schema activity and with the green solid line, we depict the project activity.

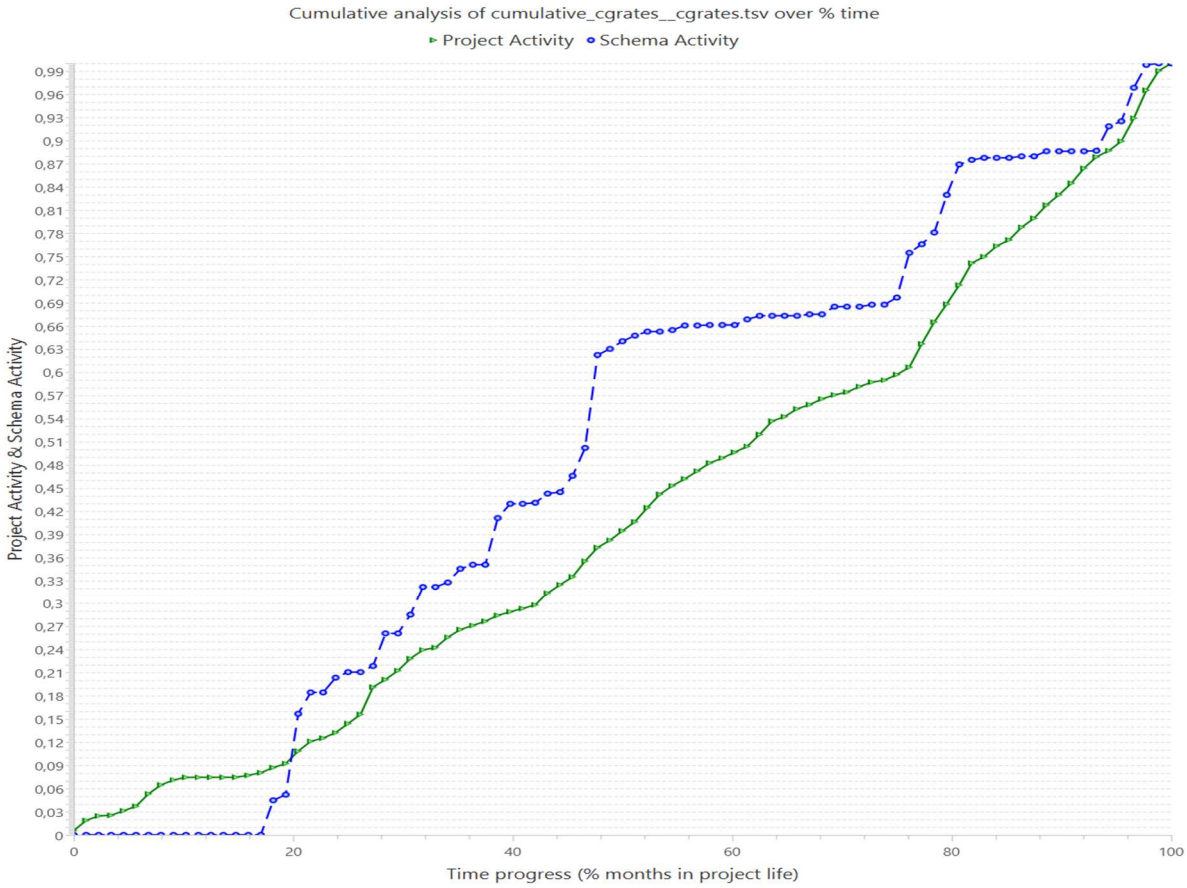


Figure 4.3 Line chart image of the cumulative analysis exported from the EvolutionChartExporter.

Finally, Figure 4.4 shows the file format of the exported tsv file from the EvolutionChartExporter. As we can see, there are six columns, we explained in section 4.1.1 what each column is.

	Month		SchActivity	PrjActivity	cumulPtime	cumulSchActivity	cumulPrjActivity
2	0	0	20	0.0	0.0	0.039761431411530816	
3	1	0	14	0.034482758620689655		0.0	0.06759443339960239
4	2	115	57	0.06896551724137931	0.39792387543252594		0.18091451292246521
5	3	138	97	0.10344827586206896	0.8754325259515571		0.37375745526838966
6	4	0	0	0.13793103448275862	0.8754325259515571		0.37375745526838966
7	5	0	5	0.1724137931034483	0.8754325259515571		0.3836978131212724
8	6	0	0	0.20689655172413793	0.8754325259515571		0.3836978131212724
9	7	0	2	0.2413793103448276	0.8754325259515571		0.38767395626242546
10	8	0	0	0.27586206896551724	0.8754325259515571		0.38767395626242546
11	9	0	0	0.3103448275862069	0.8754325259515571		0.38767395626242546
12	10	0	0	0.3448275862068966	0.8754325259515571		0.38767395626242546
13	11	0	0	0.3793103448275862	0.8754325259515571		0.38767395626242546
14	12	18	99	0.41379310344827586	0.9377162629757785		0.584493041749503
15	13	18	125	0.4482758620689655	0.9999999999999999		0.8330019880715707
16	14	0	0	0.4827586206896552	0.9999999999999999		0.8330019880715707
17	15	0	6	0.5172413793103449	0.9999999999999999		0.84493041749503

Figure 4.4 File format of the exported cumulative file.

4.2.2 Testing the cumulative analysis of EvolutionChartExporter

To ensure that the cumulative algorithm is correctly implemented into our software, we had to write and run some tests. To do that, we made two types of tests, same as the first implementation of EvolutionChartExporter in chapter 3.
For the first type of test, we created two files, representing the input of the cumulative analysis files, one matching the commitSummary format file and the other matching the MonthlySchemaStats format. After that, we compute manually the results of the cumulative analysis and created an expected cumulative result file. In the end, we created the ComputeCumulativeTest JUnit test in java to read our two test files, export the cumulative tsv file and compare it with the expected.
For the second type of test, we made a visual check into some randomly selected projects to find possible mistakes. We made visual tests to the exported tsv files and the exported line chart images.

4.3 Answering the research questions

At the very beginning of this thesis, we introduced two main research questions. In this section, we will present the process we followed and our findings and results. As mentioned in Chapter 1, these two researcher questions are:

- ***Research Question 1:*** What percentage of the projects demonstrates a "hand-in-hand" co-evolution, where the schema evolution heartbeat closely follows the heartbeat of the project?
- ***Research Question 2:*** What percentage of projects demonstrates the 80-20 rule reported in the literature [3], i.e., 80% of the schema evolution activity was obtained in the first 20% of the time?

In the following sections, we will analyze each question, and what we tried to better understand by these two questions.

4.3.1 Research question 1: What percentage of the projects demonstrates a "hand-in-hand" schema and source code co-evolution?

The first research question tries to understand if and how much the schema evolution closely follows the projects' evolution. To answer that, we used the files we exported from the EvolutionChartExporter tool and we presented in the previous sections. To measure the percentage of each project that fulfils the prerequisites, "hand-in-hand" co-evolution, we used two range windows, ±5% and ±10%. We created two python scripts to measure these and plot bar charts for each taxon and an overall chart.

The algorithm counts for each month the distance between the schema evolution and the project evolution, and divide it by the projects' life to measure the "hand-in-hand" percentage co-evolution. The algorithm is presented below.

Algorithm Computation of "hand-in-hand" co-evolution

```
1:  for each taxon txn:
2:      for each project prj:
3:          prjLife = getProjectLife();          /* a list of months */
4:          cnt10 = 0;
5:          cnt5 = 0;
6:          for each month m in prjLife:
7:              if ( abs(cumulPrjActivity – cumulSchActivity) ≤ 0.1):
8:                  cnt10++;
9:                  if ( abs(cumulPrjActivity – cumulSchActivity) ≤ 0.05):
10:                     cnt5++;
11:                 end if
12:             end if
13:         end for
14:     handInHand10perc = (100*cnt10)/prjLife;
15:     handInHand5perc = (100*cnt5)/prjLife;
16:     end for
17: end for
```

Algorithm 4.2 Computation of "hand-in-hand" co-evolution.

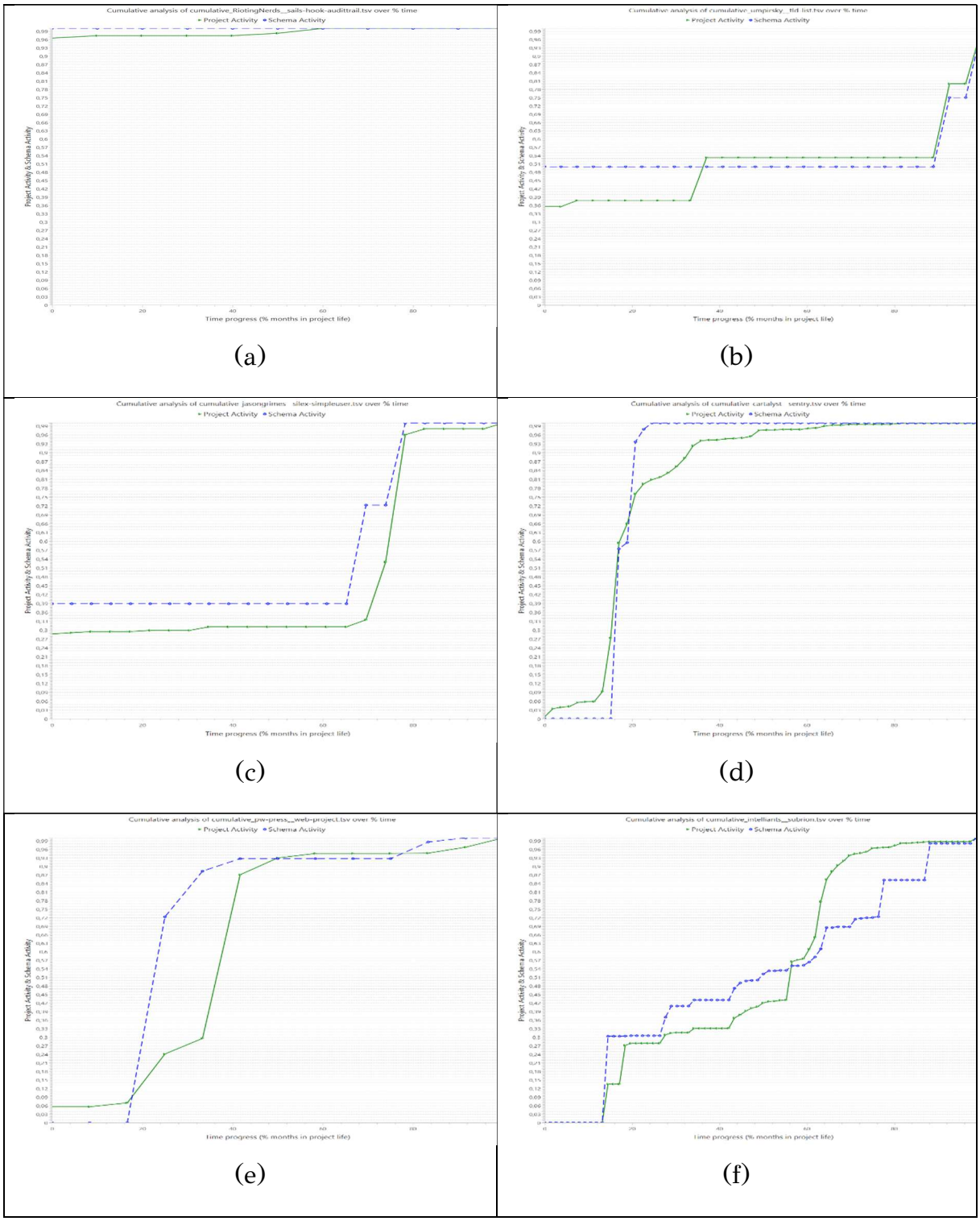


Figure 4.5 Line charts were "hand-in-hand" co-evolution is applied for the taxa: (a) FROZEN, (b) ALMOST FROZEN, (c) FocusedShot n FROZEN, (d) MODERATE, (e) FocusedShot n LOW, (f) ACTIVE.

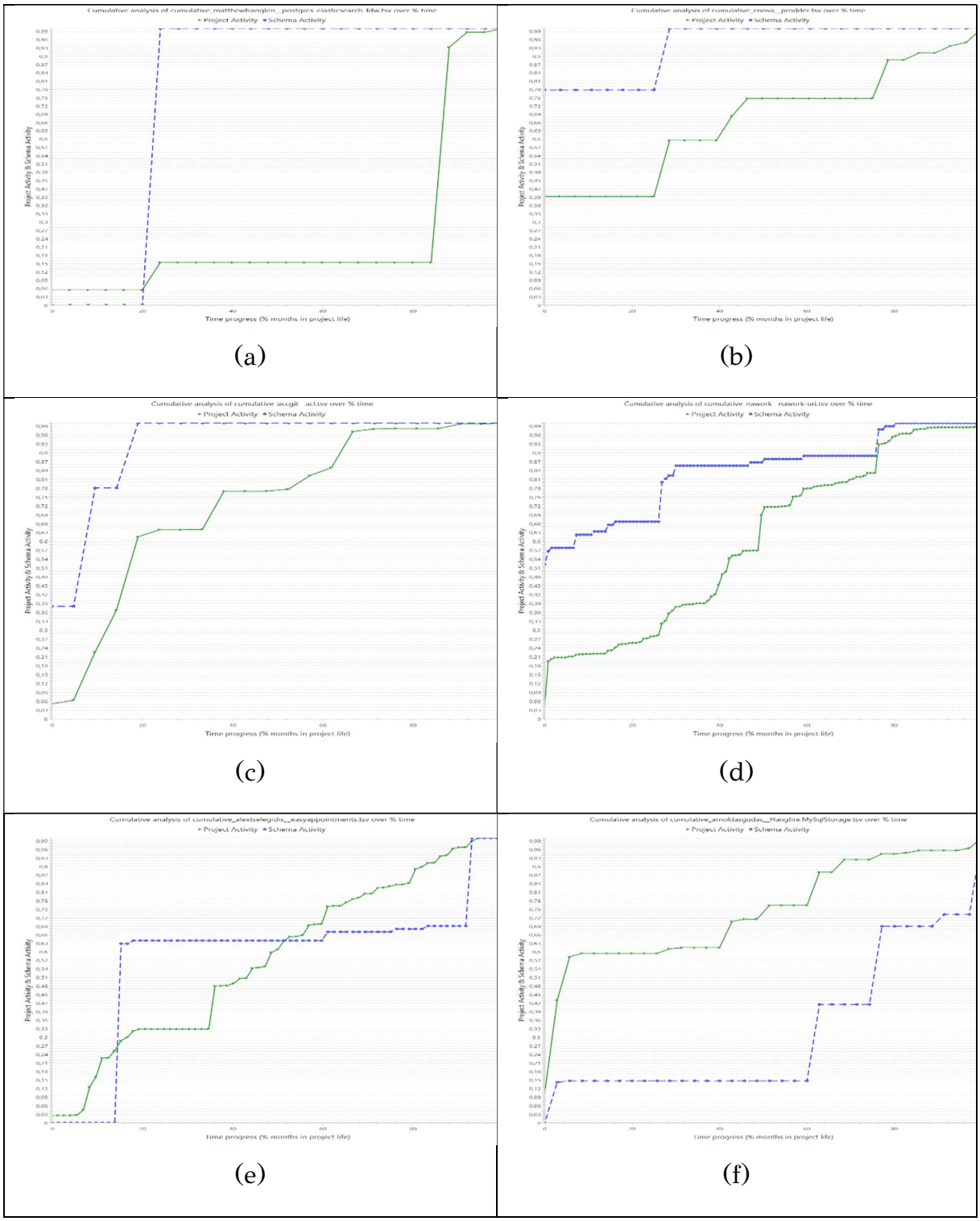


Figure 4.6 Line charts were "hand-in-hand" co-evolution is not applied for the taxa: (a) FROZEN, (b) ALMOST FROZEN, (c) FocusedShot n FROZEN, (d) MODERATE, (e) FocusedShot n LOW, (f) ACTIVE.

In Figure 4.5 we quote as an example, a line chart for each taxon, where the “hand-in-hand” schema and source code co-evolution is applied in a large percentage of the project’s life. The line charts are extracted from the EvolutionChartExporter.
In Figure 4.6 we quote as an example, a line chart for each taxon, where the “hand-in-hand” schema and source code co-evolution is not applied.

We grouped the projects into five ‘buckets’, each ‘bucket’ shows the percentage of time in which the project and schema evolution is "hand-in-hand". These ‘buckets’ are: [0%-20%) – [20%-40%) – [40%-60%) – [60%-80%) – [80%-100%]. So, for a specific project, the schema cumulative percentage line is hand-in-hand with its project cumulative percentage line in 55% of the months, the project is allocated to the 40%-59% bucket. We do this assignment for each project and then, we can count, (for each taxon and overall) what fraction of the population belongs to each bucket.
To help us to better understand and extract results, we visualized these ‘buckets’ into bar charts and also created two tables for the ±5% and ±10% windows range. In bar charts are shown, in Y-axis the number of projects that belongs to each bucket and in X-axis is the five ‘buckets’.
In Figure 4.7, we can see the charts for the "hand-in-hand" co-evolution we found for the ±5% window range.

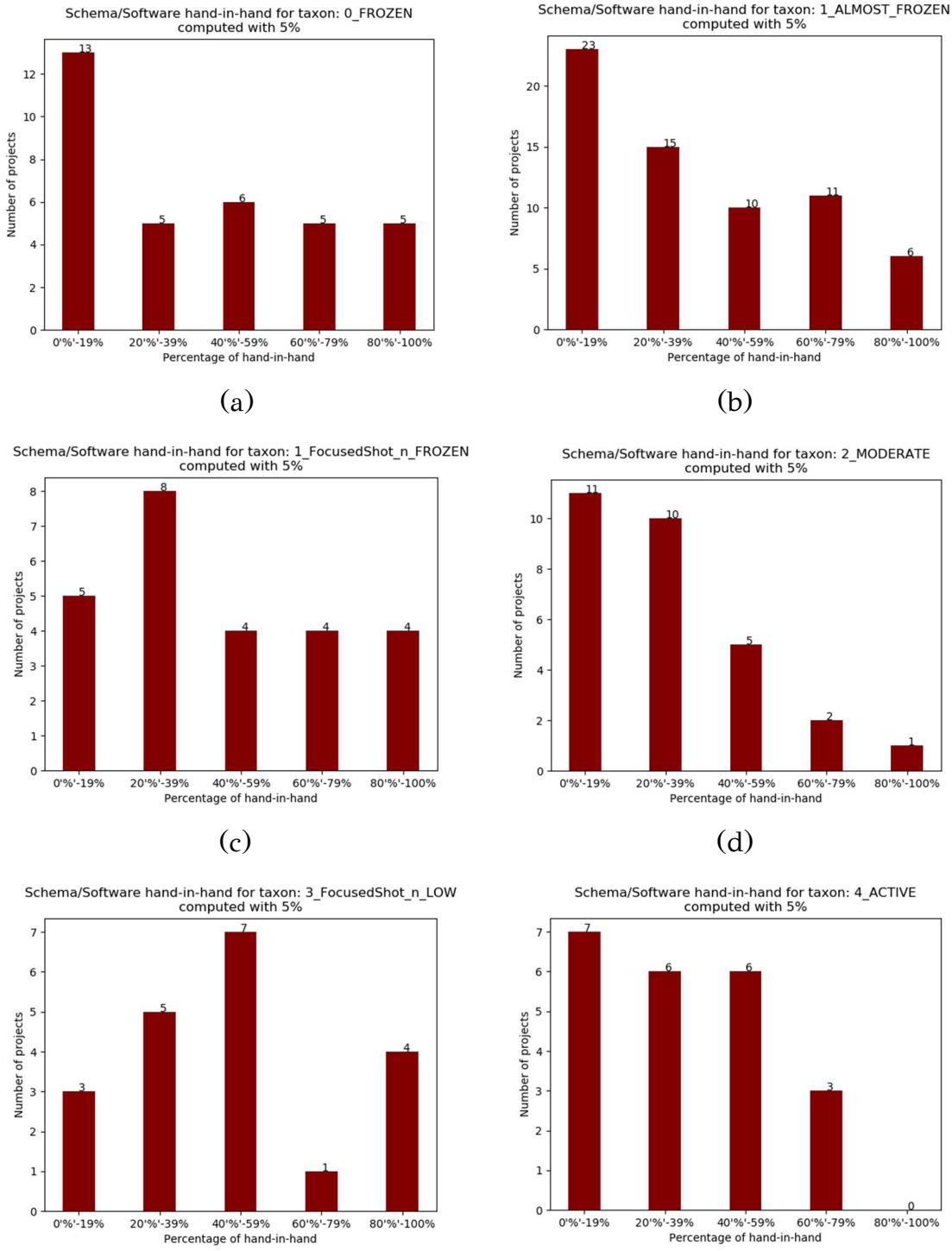
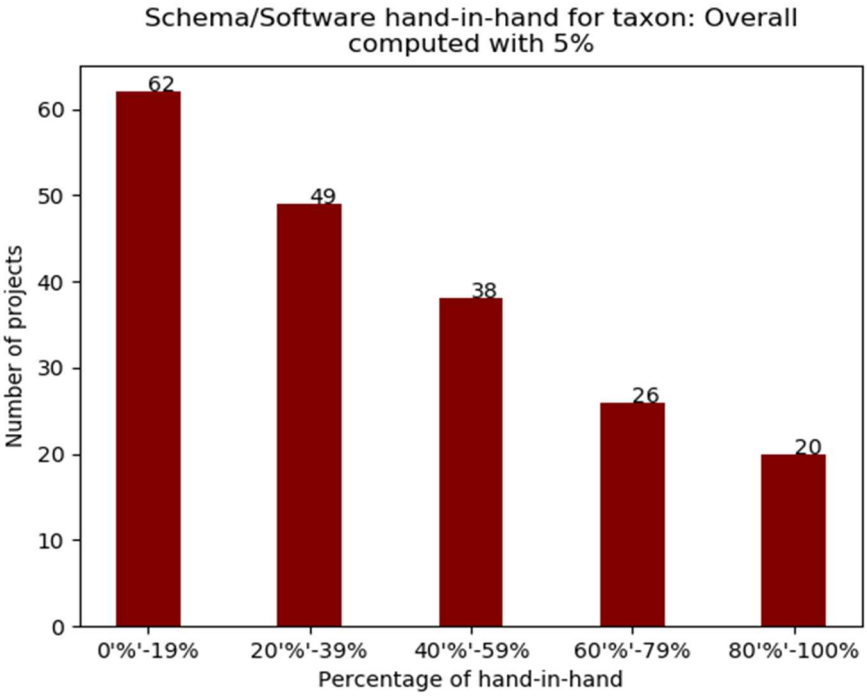


Figure 4.7 "Hand-in-hand" co-evolution for ±5% range for the taxa: (a) FROZEN, (b) ALMOST FROZEN, (c) FocusedShot n FROZEN, (d) MODERATE, (e) FocusedShot n LOW, (f) ACTIVE.

Figure 4.8 shows the overall measures for ±5% range.



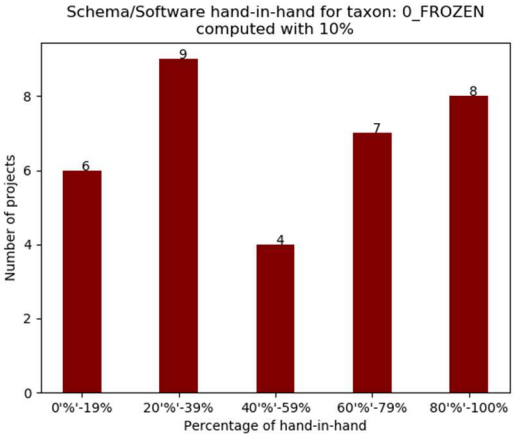
(a)

Table with 5% hand-in-hand

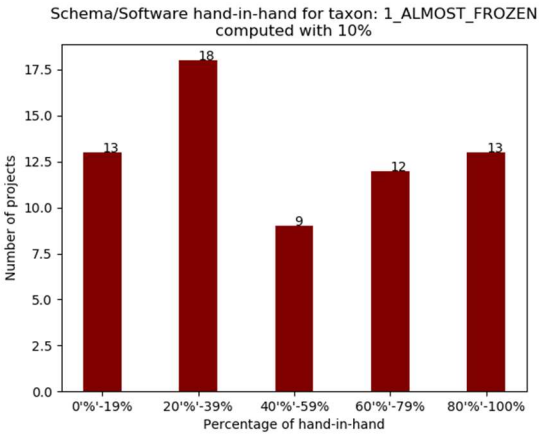
Taxon	0%-19%	20%-39%	40%-59%	60%-79%	80%-100%
0_FROZEN	13	5	6	5	5
1_ALMOST_FROZEN	23	15	10	11	6
1_FocusedShot_n_FROZEN5	5	8	4	4	4
2_MODERATE	11	10	5	2	1
3_FocusedShot_n_LOW	3	5	7	1	4
4_ACTIVE	7	6	6	3	0
Overall	62	49	38	26	20

(b)

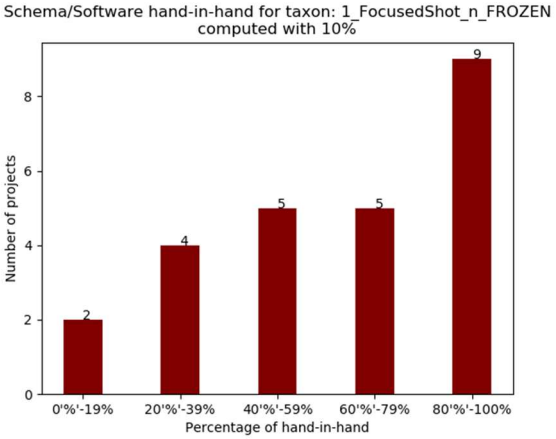
Figure 4.8 Overall "hand-in-hand" co-evolution for ±5% range: (a) Overall bar chart, (b) Table with each taxon and overall.



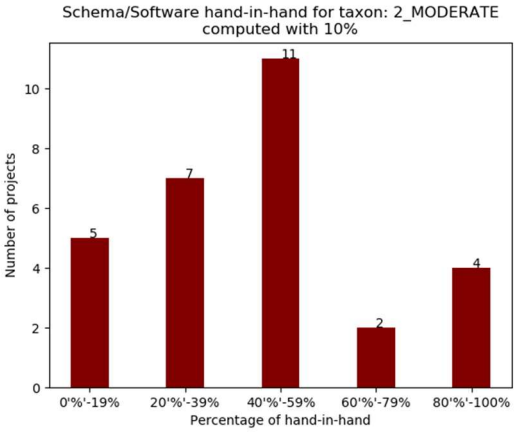
(a)



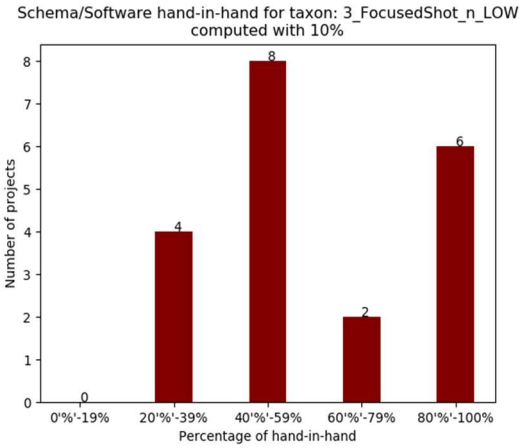
(b)



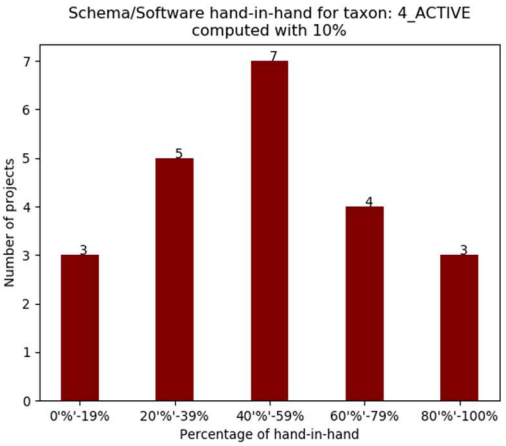
(c)



(d)



(e)



(f)

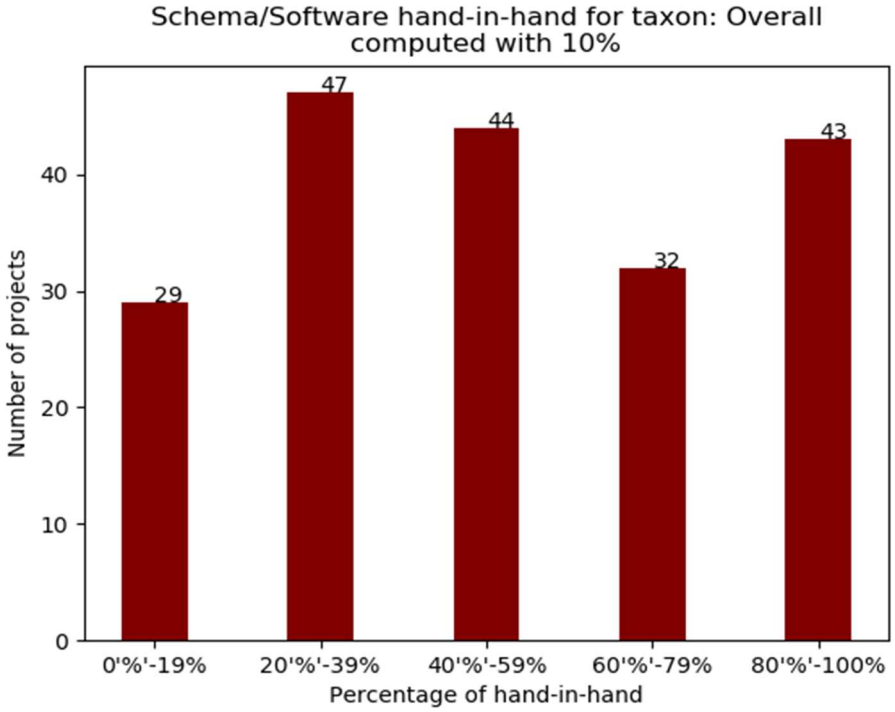
Figure 4.9 "hand-in-hand" co-evolution for ±10% range for the taxa: (a) FROZEN, (b) ALMOST FROZEN, (c) FocusedShot n FROZEN, (d) MODERATE, (e) FocusedShot n LOW, (f) ACTIVE.

As we can see, for the window of ±5% range, the "hand-in-hand" co-evolution decreases over time, this is shown clearly in the overall results. Also, we can conclude that the four first taxa, that are less active, most of the projects have a very small percentage of "hand-in-hand" co-evolution. On the other hand, the most active taxa, the FocusedShot n LOW and ACTIVE tend to have a small or average percentage of "hand-in-hand" co-evolution.

We also used a ±10% window range, double the previous range. We expect the life percentage of the "hand-in-hand" to move to the ‘right’ for all the taxa.

In Figure 4.9, we can see the results for each taxon.

Figure 4.10 shows the overall measures for ±10% range.



(a)

Table with 10% hand-in-hand

Taxon	0%-19%	20%-39%	40%-59%	60%-79%	80%-100%
0_FROZEN	6	9	4	7	8
1_ALMOST_FROZEN	13	18	9	12	13
1_FocusedShot_n_FROZEN5	2	4	5	5	9
2_MODERATE	5	7	11	2	4
3_FocusedShot_n_LOW	0	4	8	2	6
4_ACTIVE	3	5	7	4	3
Overall	29	47	44	32	43

(b)

Figure 4.10 Overall "hand-in-hand" co-evolution for ±10% range: (a) Overall bar chart, (b) Table with each taxon and overall.

As we can, the life percentage that a project and schema evolution is "hand-in-hand" increased. We can observe that more than 1/5 of the projects are "hand-in-hand" co-evolving almost completely.

4.3.2 Research question 2: how premature is schema evolution completion?

The second research question is a result reported in the literature [3], the “80-20 rule”, suggesting that 80% of the schema changes are completed in the first 20% of the projects’ life. We wanted to see if this rule applies to a large number of projects. To answer that, we used the exported files from the EvolutionChartExporter. We created again two scripts to count and plot the results. The first script counts when in the projects’ life, in percentage, the schema activity reaches a specific percentage. Using this script, we create, for each taxon, a file that has the project name, the projects’ life in months and projects’ life percentage, we decided to broaden the research so we took four cases when the schema activity reaches 50%, 75%, 80% and 100%. To find these percentages, we implemented an algorithm that is introduced in Algorithm 4.3.

Algorithm Computation of 80-20 rule, and more

```
1:  for each taxon txn:
2:      for each project prj:
3:          time = [0, 0, 0, 0;                    // for [50%, 75%, 80%, 100%]
4:          for each month m in prjLife:
5:              if (cumulSchActivity = 1.0):
6:                  time[3] = cumulPTime;
7:              else if (cumulSchActivity ≥ 0.8):
8:                  time[2] = cumulPTime;
9:              else if (cumulSchActivity ≥ 0.75):
10:                 time[1] = cumulPTime;
11:             else if (cumulSchActivity ≥ 0.5):
12:                 time[0] = cumulPTime;
13:             end if
14:         end for
15:     end for
16: end for
```

Algorithm 4.3 Computation of 80-20 rule, and more

Using this algorithm, we found at which percentage of the time, each project reached 50%, 75% 80% and 100% of schema activity. Moreover, based on the findings, we also created a script to visualize these measures.

Figure 4.11 depicts several instances of our measurements as a bar chart. In each bar chart, we observe the following characteristics:

- The horizontal axis refers to the percentage of schema activity measured.
- The series refers to the range of project lifetime within which this activity was obtained (again as a percentage of a total lifetime).
- The vertical axis counts how many projects refer to this combination of what percentage of evolutionary activity was completed within this percentage of the time.

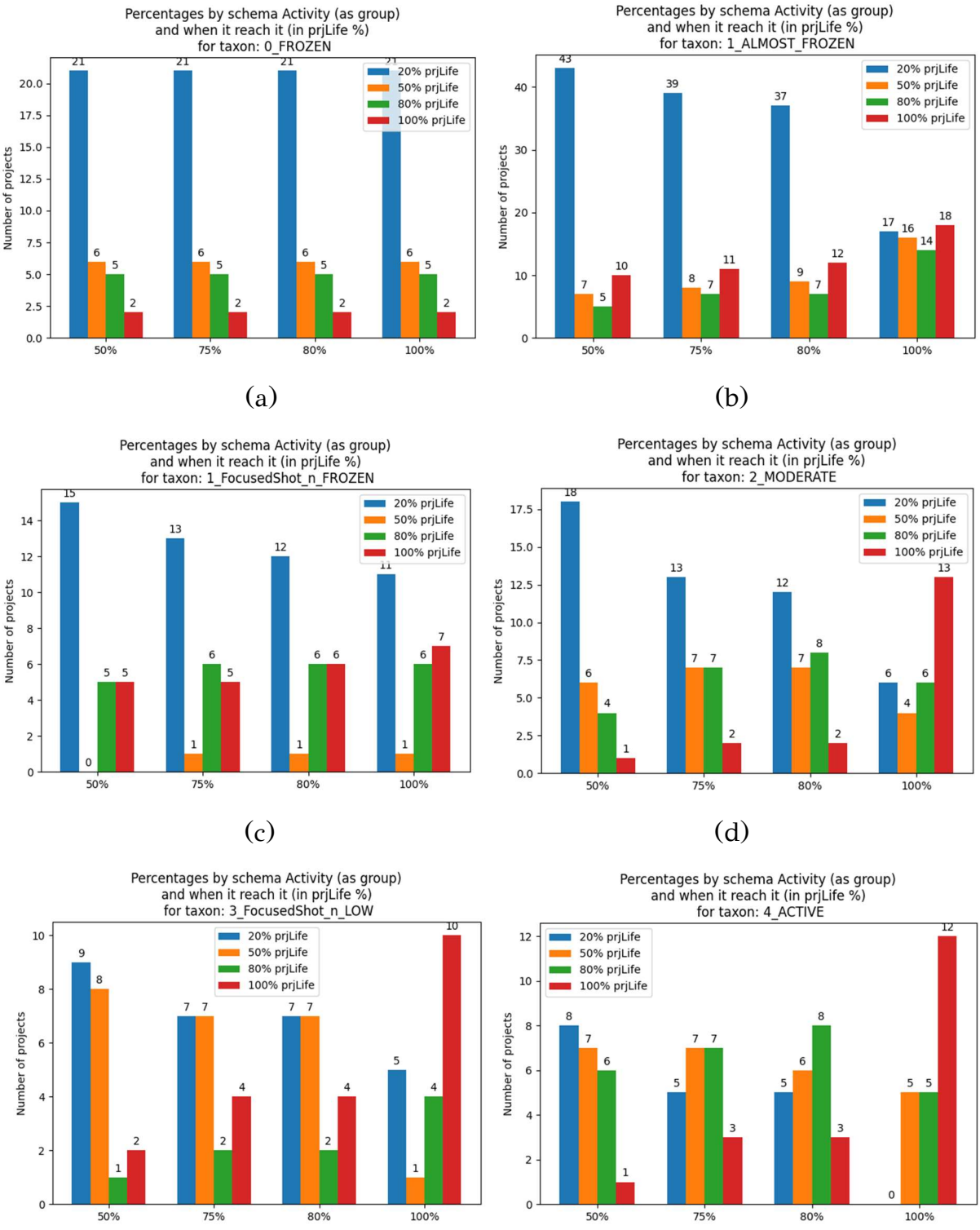
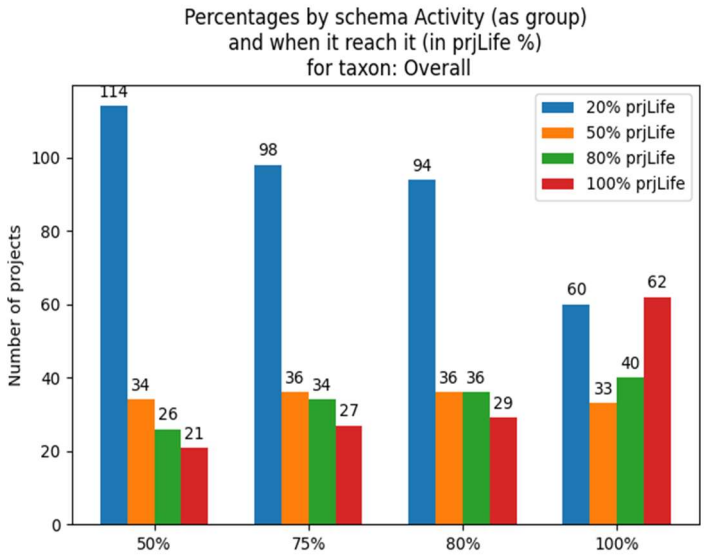


Figure 4.11 When (in %) each project reached a specific schema activity for the taxa: (a) FROZEN, (b) ALMOST FROZEN, (c) FocusedShot n FROZEN, (d) MODERATE, (e) FocusedShot n LOW, (f) ACTIVE.

For example, in Figure 4.11 (b), for the x-axis value of 80% (meaning that we measure when 80% of evolutionary activity was reached), we see that out of the 65 projects, 37 of them completed this 80% of activity within 20% of their project lifetime (the blue bar), 9 of them completed this 80% of activity between 21%-50% of their lifetime, 7 of them between 51% - 80% of their lifetime, and 12 of them between 81%-100% of their lifetime. Figure 4.11 shows the results from our analysis for each taxon.

In Figure 4.12, we can see the overall results and a table with all results for each taxon and the overall.



(a)

Number of projects that reached x% of schema activity, and when

Taxon	Project Life	Schema Activity 50%	Schema Activity 75%	Schema Activity 80%	Schema Activity 100%
0_FROZEN	• 20% • 50% • 80% • 100%	• 21 • 5 • 5 • 2	• 21 • 6 • 5 • 2	• 21 • 6 • 5 • 2	• 21 • 6 • 5 • 2
1_ALMOST_FROZEN	• 20% • 50% • 80% • 100%	• 43 • 7 • 5 • 10	• 39 • 8 • 7 • 11	• 37 • 9 • 7 • 12	• 17 • 16 • 14 • 18
1_FocusedShot_n_FROZEN	• 20% • 50% • 80% • 100%	• 15 • 0 • 5 • 5	• 13 • 1 • 6 • 5	• 12 • 1 • 6 • 6	• 11 • 1 • 6 • 7
2_MODERATE	• 20% • 50% • 80% • 100%	• 18 • 6 • 4 • 1	• 13 • 7 • 7 • 2	• 12 • 7 • 8 • 2	• 6 • 4 • 6 • 13
3_FocusedShot_n_LOW	Task 5 • 20% • 50% • 80% • 100%	• 9 • 8 • 1 • 2	• 7 • 7 • 2 • 4	• 7 • 7 • 2 • 4	• 5 • 1 • 4 • 10
4_ACTIVE	• 20% • 50% • 80% • 100%	• 8 • 7 • 6 • 1	• 5 • 7 • 7 • 3	• 5 • 6 • 8 • 3	• 0 • 5 • 5 • 12
Overall	• 20% • 50% • 80% • 100%	• 114 • 34 • 126 • 21	• 98 • 36 • 34 • 27	• 94 • 36 • 36 • 29	• 60 • 33 • 40 • 62

(b)

Figure 4.12 Overall counting of when each project reached a specific schema activity: (a) Overall bar chart, (b) Table with each taxon and overall.

It is important to understand that each group is computed separate from the other, all projects appear in every group, in the same bar on in another of the same group. For example, a project that in its 20% of life has a schema activity of 76%, is counted in both 50% and 75% groups in the same bar. The sum of the projects in each group is equal to the number of projects in the taxon or all, for the overall.
In our bar charts, the 80-20 rule is represented by the blue bar (the first 20% of the project's life) in the 80% group. We can observe that the rule, applies to the overall projects, almost half of them. We can also observe that this rule happens more often in the none so active projects.

CHAPTER 5

CONCLUSION AND FUTURE WORK

The final chapter of this thesis summarizes the major findings of our study, we outline the research questions made in the introductory chapter and we propose potential future work.

5.1 Conclusions

The aim of this thesis was to study the schema and software co-evolution. This research field is still in its early stages. The analysis of the history of a project hides lots of difficulties, is a difficult process to automate and a very time-consuming procedure to make manually. This thesis used a large collection of projects and their history and extorted statistical results.

Firstly, we studied the relevant researches made to the field of schema and software co-evolution. Then we made a manual analysis of six randomly selected projects, with the expectation to better understand what changes are made during the project life, why, where and how developers affect the schema and software. When these changes are most likely to take place and who is usually making changes to the schema.

Finally, we answered two research questions, these questions are: (a) What percentage of the projects demonstrates a "hand-in-hand" co-evolution, where the schema evolution heartbeat closely follows the heartbeat of the project? and (b) What percentage of projects demonstrates the 80-20 rule reported in the literature [3]. I.e., 80% of the schema evolution activity was obtained in the first 20% of the time? In our research, we studied also 50%, 75% and 100%. For the first research question

we found that overall, 1/5 of the projects are co-evolving hand-in-hand. For the second research question, we found that the 80-20 rule is not negligible and was applied in half of our projects.
We also presented a tool we made, called EvolutionChartExporter to help us compute the required metrics and visualize these for a better understanding.

5.2 Future work

In follow-up work, one can better define the source code activity and extract the actual software changes. Also, a deeper investigation and automation of schema activity extraction can possibly give better grouping, taxa, and as a result of these a better understanding of the schema and software co-evolution. Finally, we created a new tool, the EvolutionChartExporter, in future work, someone can add new features and metrics.

REFERENCES

[1] Dien-Yen Lin, Iulian Neamtiu [YeNe09]. Collateral Evolution of Applications and Databases, 2009.

[2] Shengfeng Wu, Iulian Neamtiu [WuNe11]. Schema Evolution Analysis for Embedded Databases, 2011.

[3] Dong Qiu, Bixin Li, Zhendong Su [QuLS13]. An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications, 2013.

[4] Panos Vassiliadis [Vass21]. Profiles of Schema Evolution in Free Open-Source Software Projects. 37th IEEE International Conference on Data Engineering (ICDE '21), Chania, Crete, Greece, 19-22 April 2021

SHORT BIOGRAPHICAL SKETCH

Fation Shehaj was born in Fieri, Albania. In 2017, he received his Diploma in Computer Science and Engineering from the University of Ioannina. After fulfilling his military obligations, we worked as a freelance developer for a local company in Rhodes. In 2019, he started his graduate studies at the Department of Computer Science & Engineering at the University of Ioannina while working as a mobile developer in the R&D hub of a private company specialized in medical solutions.