Manual analysis of uses cases of source and schema co-evolution

Fation Shehaj, Panos Vassiliadis

1. Introduction

In this report, we make a deep analysis of six randomly selected projects towards obtaining a better understanding of schema and source code co-evolution. To this end, we manually examined the history of the commits for these projects towards extracting and understanding the schema changes and the code changes triggered from the schema changes. After presenting the details of the individual cases, we also summarize our findings from our deep analysis of the six projects and report on our efforts to locate patterns from the studied projects. We are also going to present a tool we created to examine visually the occurrence of code and schema changes over time, we named that tool *EvolutionChartExporter* (ECE).

2. Manual Analysis

In this Section, we present our manual examination of six randomly selected projects. For our study, we used the dataset of 195 schema histories of [Vass21], which also separates the 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 criteria used to classify these projects were firstly introduced in [Vass21] and is shown in Figure 1.

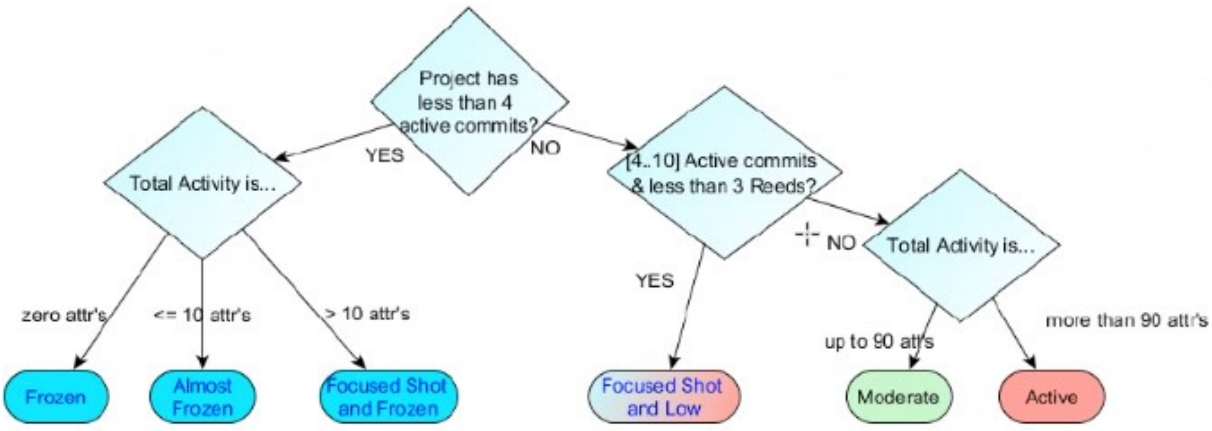


Figure 1 Taxa of Schema Evolution for FOSS Projects [Vass21]

In our deliberations, we selected a set of 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 indeed observed, that the number of commits to examining was increasing rapidly from one taxon to the other.

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 data that we extracted from the git history of the respective project; their semantics are: The Column *Date* contains the date of a specific commit in GitHub and the column *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. 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 overall, we checked only the commits with a schema change and we decided to use a *'window' of ±3 commits for 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 dramatically decreased the time required to do the process. 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 fixes to the source code. We do not present here the commits related only to the code as we are interested in schema and source code coevolution.

"No-Change" Annotations. Each of our case studies comes with a table that characterize the commits related to the schema changes. When there are no changes from one commit to another, we will write *no change.* The semantics of no change are as follows:

- When *no change* is written at the column for the source code, it means that in this commit, there were no changes to the code related to the database changes. So, the source code is the same as the previous commit with respect to its relationship to the database. This does not mean that there is no change overall (otherwise, why the purpose of a commit?), but that the change concerns parts that are irrelevant to the database. For example, if a certain commit changed the way a function computes an algorithmic result, we mark the cell of the table as *no change,* as we are not interested in that kind of changes.
- When we write *no change* at the DB column of the table, 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?
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?

2.1 In-depth study of ALMOST_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

2.1.1 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.

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.

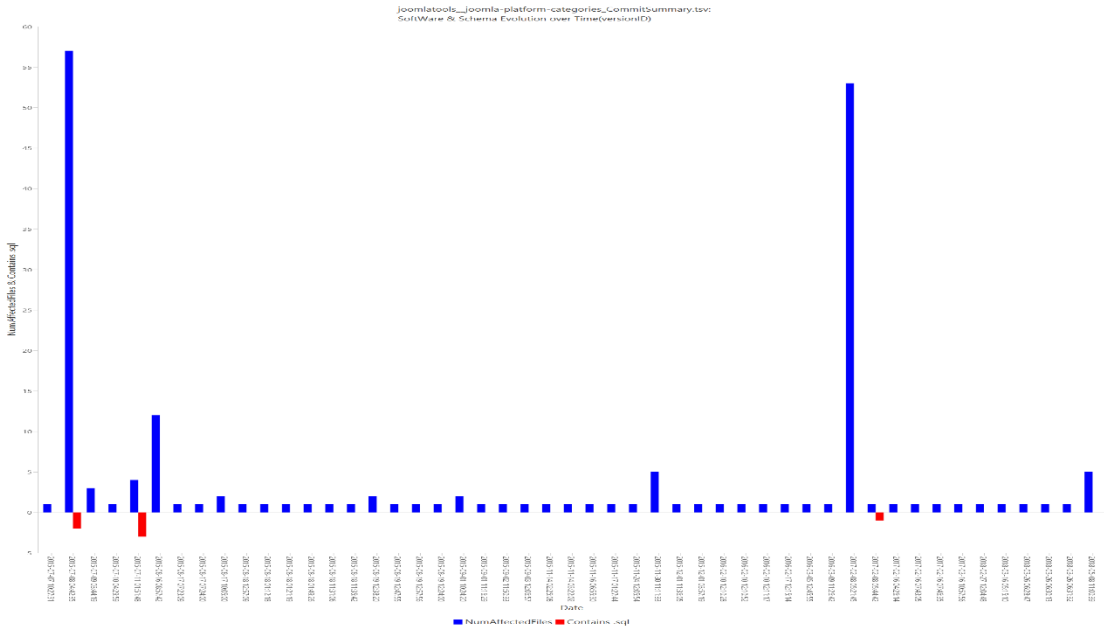


Figure 2 Schema and src commits for joomla-platform-categories

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:** Johan: 2/3 (one was the initial commit); Allan: 1/3

Table 2.1. Schema-related commits 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.

2.1.2 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 2.2 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:** umpirsky: - 1/4 (one was the initial commit); M: -1/4; Saša: - 2/4

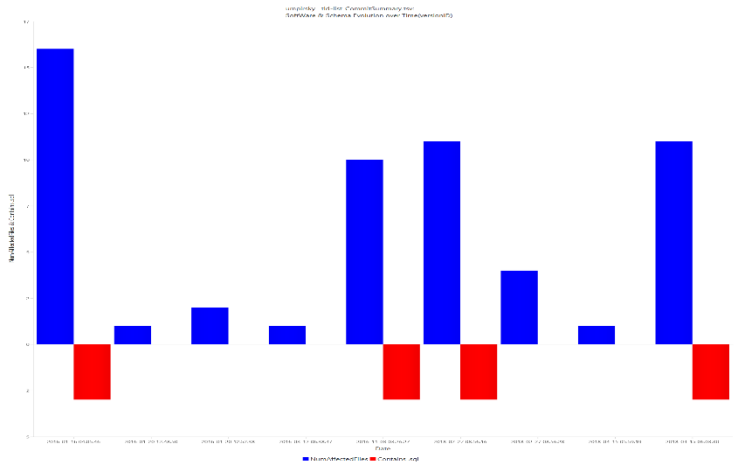


Figure 3 Schema and src commits for tld-list

2.1.3 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.

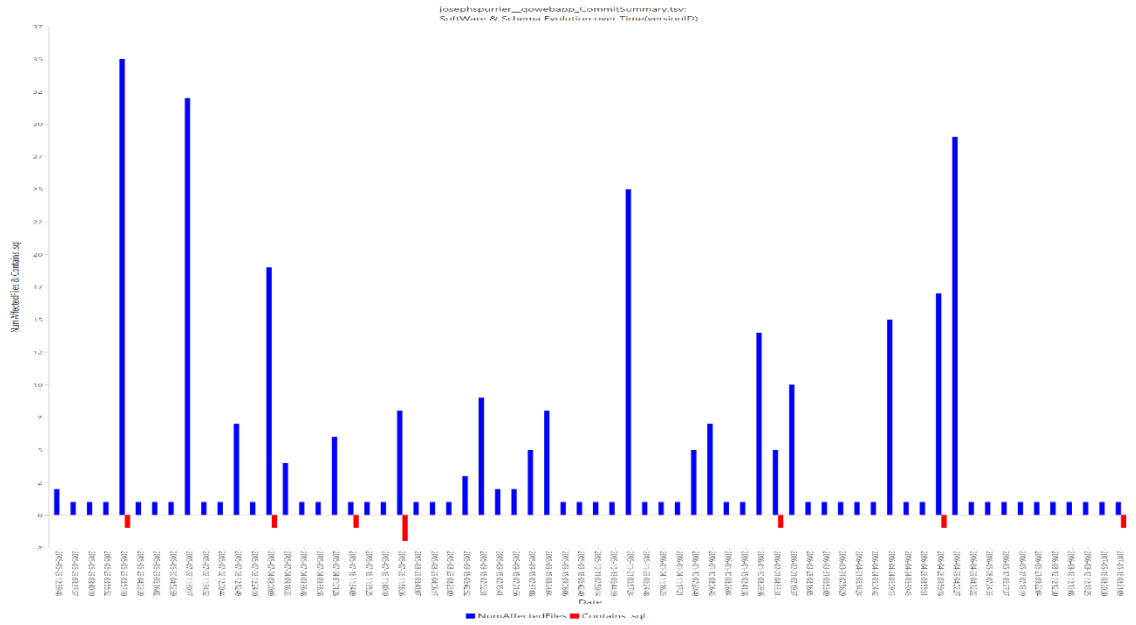


Figure 4 Schema and src commits for gowebapp

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:** Joseph: 8/9 (one was the initial commit); Shane: 1/9

Table 2.3 Commits related to the schema for the josephspurrier/gowebapp project

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

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

2.2.1 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 2.4 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`, `role`, `user`) VALUES (NULL, **2**, 1);	**DB** 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 08:33:40 +0200	Zdeněk	0	1	**DB** (NULL, 'Admin:Admin'), (NULL, 'Web:Login'), (NULL, 'Web:Web');	**DB** 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:** Zdeněk: 18/18 (one was the initial commit)

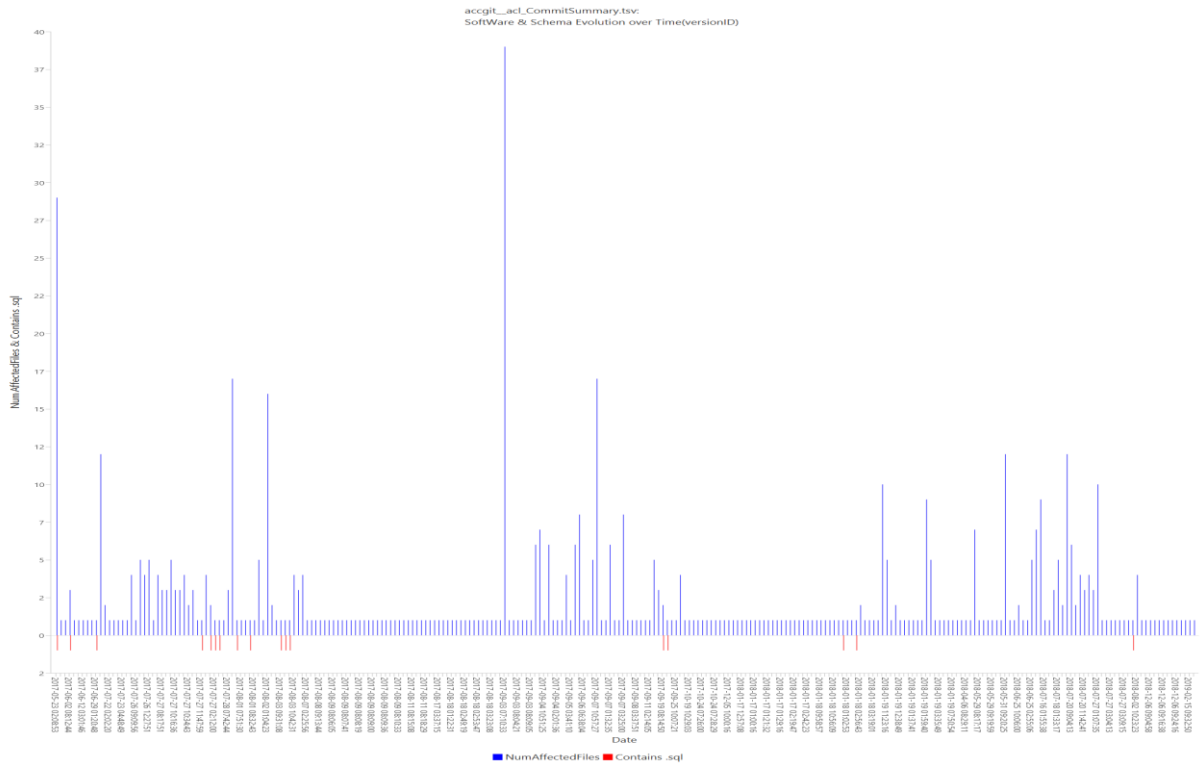


Figure 5 Schema and src commits for acl

2.2.2 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 2.5 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:** Jason: 11/15 (one was the initial commit); enyosolutions: 4/15 (commits canceled due to conflicts)

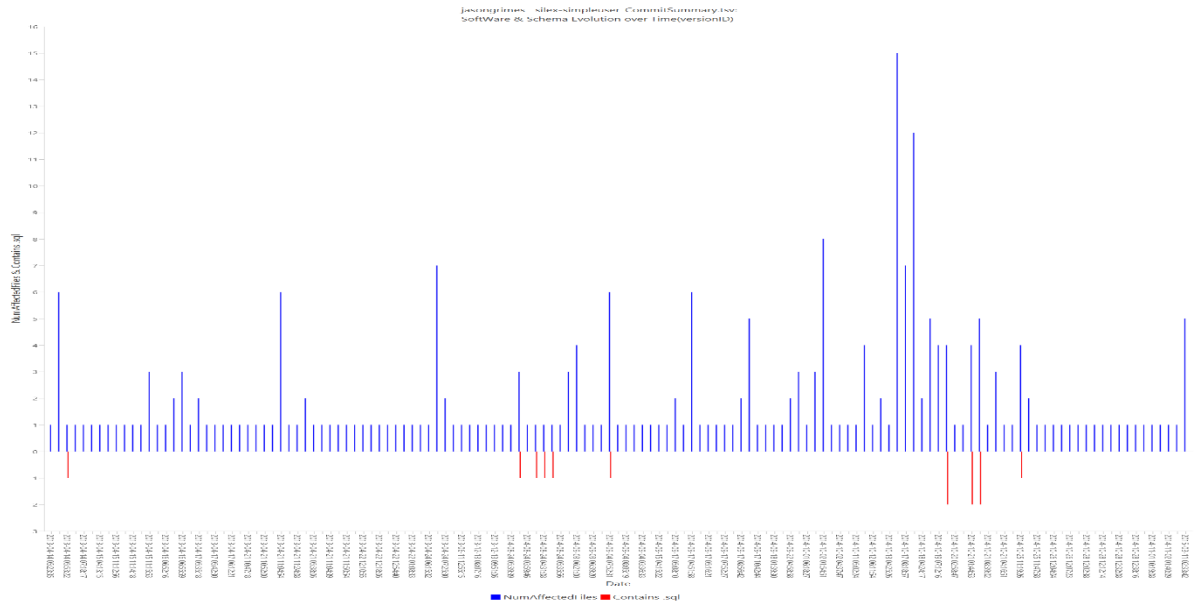


Figure 6 Schema and src commits for silex-simpleuser,

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

2.3.1 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 2.6 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 No Code	**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.DATE;

2015-11-19 12:05:42 +0530	Sanjay	3	2	**DB** Into create_tables.sql	**DB** UPDATE changesets SET bbox =

				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	ST_MakeEnvelope(min_lon, min_lat, 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	**CODE**

				No change	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_GeomFromText($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_GeomFromText($5, 4326) where id=$1';
2016-02-23 13:31:37 +0530	Sanjay	0	1	**DB** No change	**DB** changesets/post_initial.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,	**DB** Save is_unreplied boolean, add to schema

				num_changes, min_lon, min_lat, max_lon, max_lat) FROM 'csv/changesets.csv' DELIMITERS ',' CSV;	\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	**CODE**

				No change	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, changeset_count, num_changes from users where id=$1';	**CODE** Add **first_edit** to user select query, wasn't retrieved -> 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:** Sanjay 27/30 (one was the initial commit); Kushan 3/30

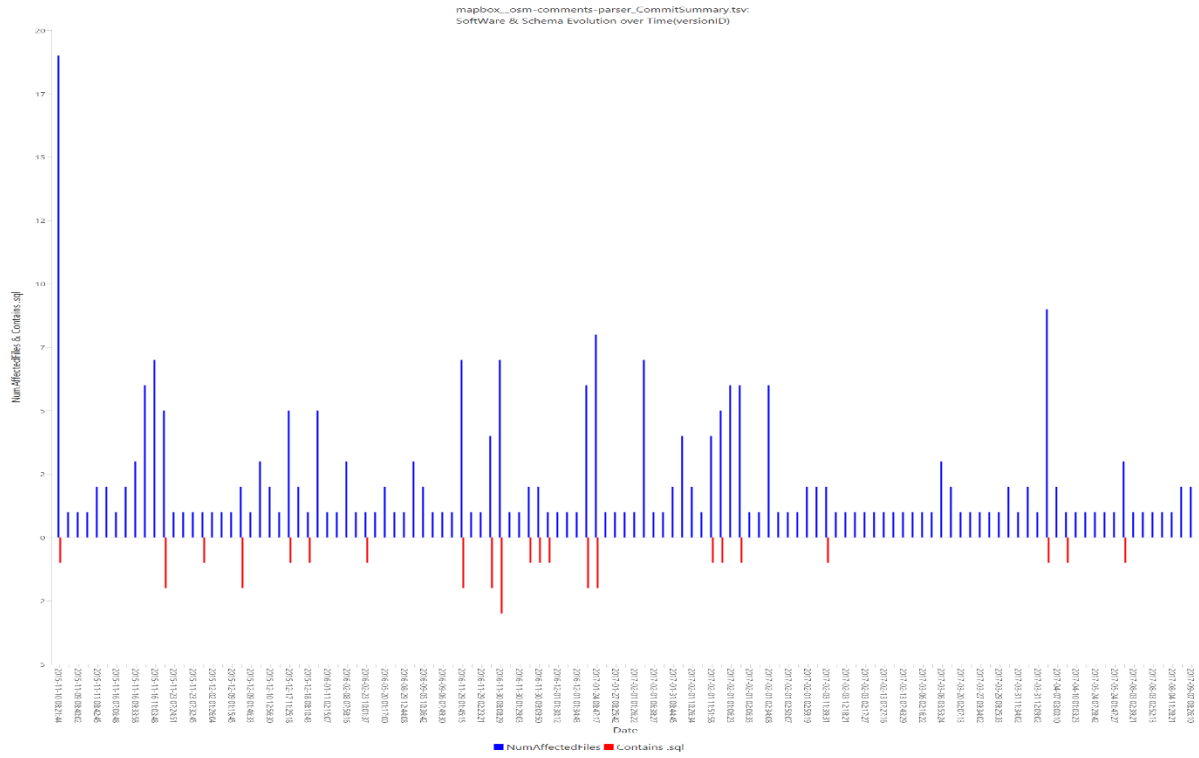


Figure 7 Schema and src commits for osm-comments-parser

3. Findings

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.

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 8, 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.

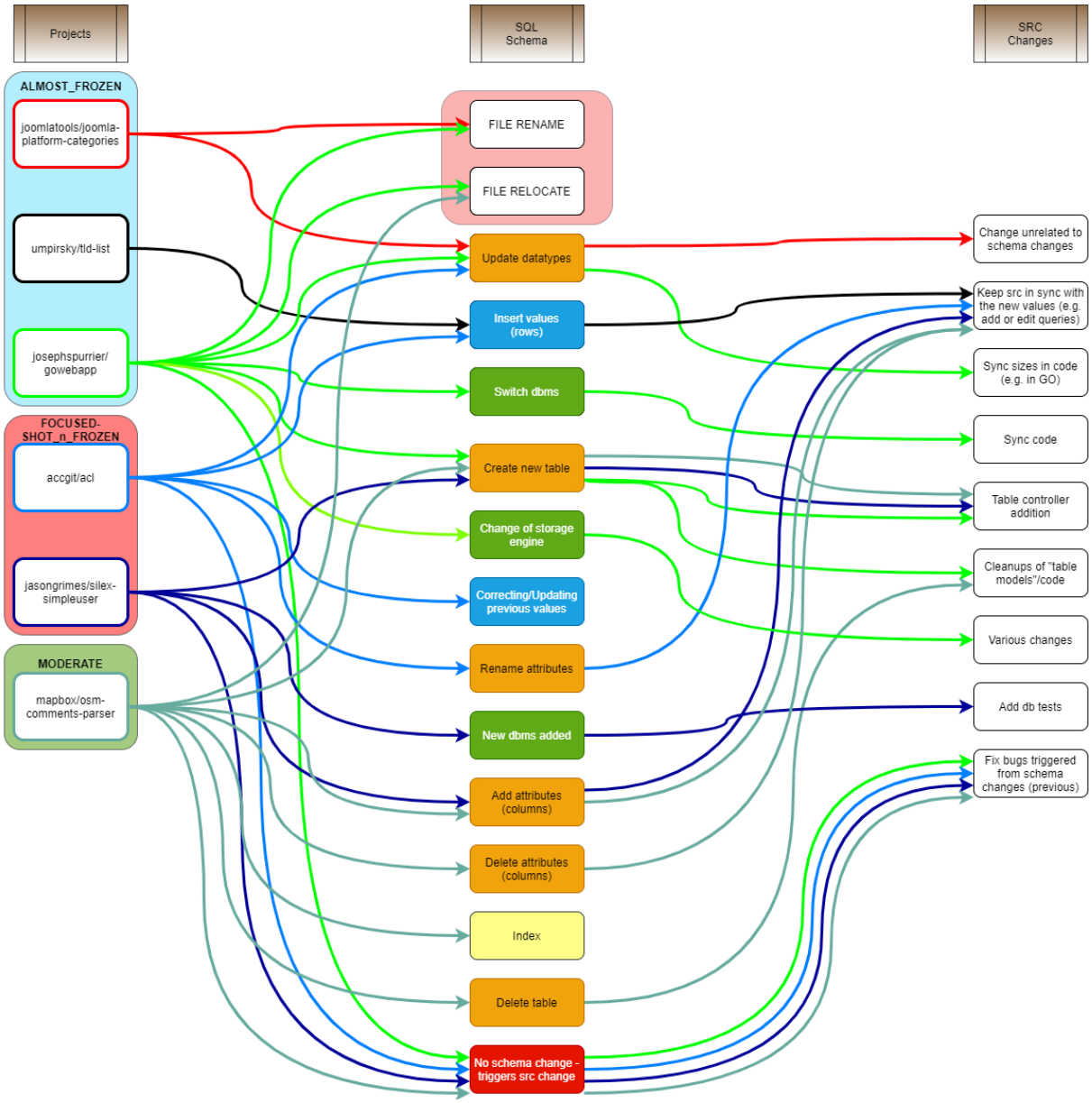


Figure 8 Analysis of schema changes per project and their impact on source code

In Figure 8, 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 at logical level (orange).
2) Change at accompanying data in the Data Definition Language File (blue).
3) Change at engine supported (green).
4) Change at accompanying database code (red).
5) Change at physical level (yellow).

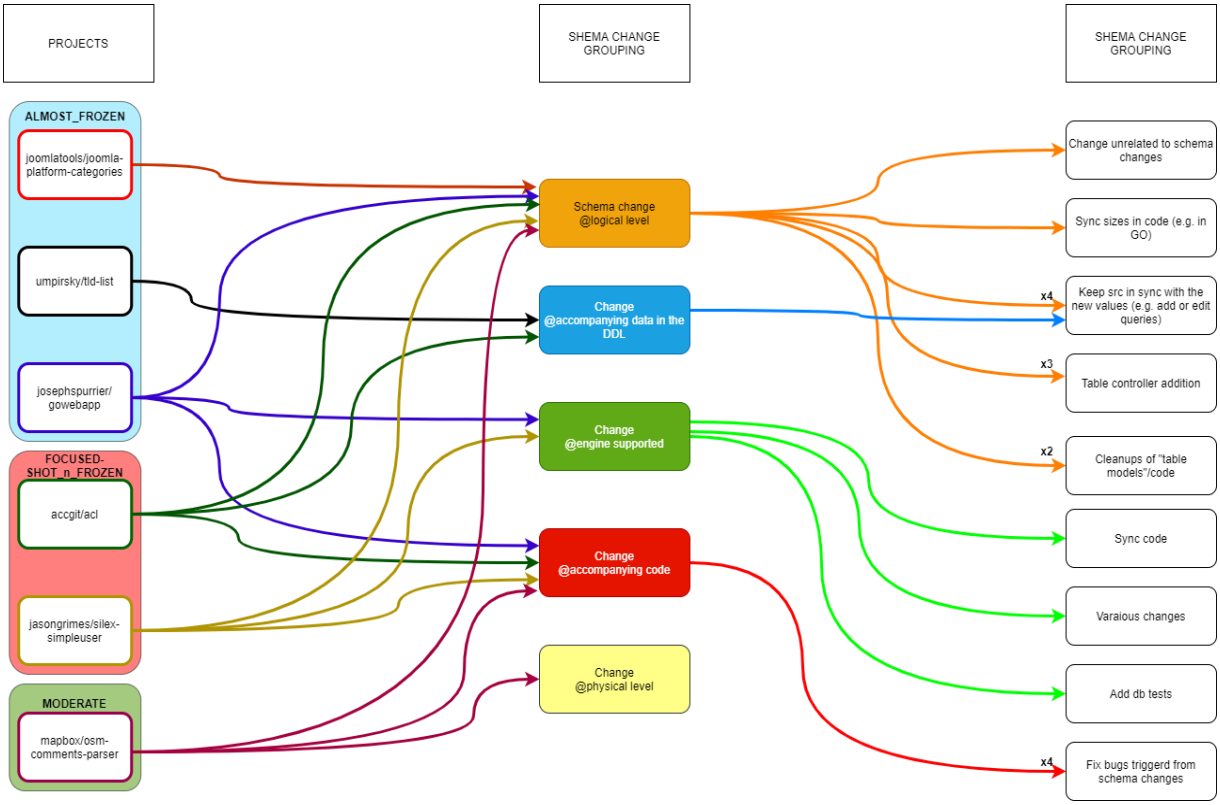


Figure 9 Grouped schema changes and their impact on source code

In Figure 9 we can clearly identify what impact has each group of schema change to the source code.

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 3.1 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.

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 3.2 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.

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 3.3 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%

4. Toolset

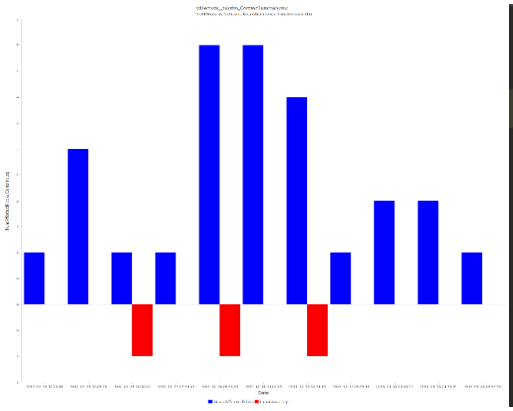
4.1 Introduction to EvolutionChartExporter

The main reason we decided to create the EvolutionChartExporter tool is to visualize the amount of code and schema changes that have been committed over time. As we mentioned, we collected the commit history for 350 projects from GitHub [Vass21]. 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.

To create the EvolutionChartExporter tool, we used as reference the chart exporter source code from *HeraclitusFire* [Hera22]. First, we will make a brief explanation of how this tool works. In the next subsection, we will analyze more 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 the design of the EvolutionChartExporter. Finally, we will mention the tests we made to evaluate this tool.

4.2 Functionality and Architecture

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 10 below shows an example: (left) the exported image from the EvolutionChartExporter and (right) 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

Figure 10 Bar chart exported from EvolutionChartExporter (left) the exported image from the EvolutionChartExporter and (right) the format of the input file

As already mentioned, we are using as the corpus from which use cases were drawn, the 195 projects of [Vass21] organized into six taxa. 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*, as already described previously. 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.

4.3 Testing

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:

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.

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.

References

[Hera22] Heraclitus Fire: A Java project to enable the processing of the history of a schema for correlations and patterns. Available at https://github.com/DAINTINESS-Group/HeraclitusFire

[Vass21] Panos Vassiliadis. Profiles of Schema Evolution in Free Open Source Software Projects.37th IEEE International Conference on Data Engineering (ICDE 2021), 19-22 April 2021, Chania, Crete, Greece.
Datasets available at: https://github.com/DAINTINESS-Group/Schema_Evolution_Datasets