Joint Source and Schema Evolution: Insights from a Study of 195 FOSS Projects

Panos Vassiliadis
Univ. Ioannina
Ioannina, Greece
pvassil@cs.uoi.gr

Fation Shehaj*
NIKI Digital Engineering
Ioannina, Greece
fation.sh94@gmail.com

George Kalampokis†
GWF MessSysteme AG
Salonika, Greece
gtkalampokis@gmail.com

Apostolos V. Zarras
Univ. Ioannina
Ioannina, Greece
zarras@cs.uoi.gr

ABSTRACT

In this paper, we address the problem of the co-evolution of Free Open Source Software projects with the relational schemata that they encompass. We exploit a data set of 195 publicly available schema histories of FOSS projects hosted in Github, for which we locally cloned their respective project and measured their evolution progress. Our first research question asks which percentage of the projects demonstrates a "hand-in-hand" schema and source code co-evolution? To address this question, we defined synchronicity by allowing a bounded amount of lag between the cumulative evolution of the schema and the entire project. A core finding is that there are all kinds of behaviors with respect to project and schema co-evolution, resulting in only a small number of projects where the evolution of schema and project progress in sync. Moreover, we discovered that after exceeding a 5-year threshold of project life, schemata gravitate to lower rates of evolution, which practically means that, with time, the schemata stop evolving as actively as they originally did. To answer a second question, on whether evolution comes early in the life of a schema, we measured how often does the cumulative progress of schema evolution exceed the respective progress of source change, as well as the respective progress of time. The results indicate that a large majority of schemata demonstrates early advance of schema change with respect to code evolution, and, an even larger majority is also demonstrating an advance of schema evolution with respect to time, too. Third, we asked at which time point in their lives do schemata attain a substantial percentage of their evolution. Although there are exceptions to the general trend, a large number of projects attracts a large percentage of their schema evolution disproportionately early with respect to their project life span. Indicatively, 98 of the 195 projects attained 75% of the evolution in just the first 20% of their project's lifetime.

1 INTRODUCTION

The proliferation of the relational model and the DBMSs built on its premises has dominated the construction of information systems that are operating on top of relational databases that both store and provide data for their operation. The essence of query answering is based on declarative queries that come with precise semantics, i.e., with the ability to return as query answers exactly what the querying user has specified as her query. To achieve that, queries are authored with respect to the names of the elements of the database schema, i.e., the internal structure of a database, which in the relational case is expressed via a set of relations, their typed attributes, and intra- and inter-relation constraints.

Schema Evolution refers to the process via which the schema changes via the addition, deletion, and update of the elements of a schema and their relationships. Whenever the schema evolves, the surrounding queries of the system are potentially affected, both syntactically and semantically. The syntactic impact is due to the fact that the queries of the surrounding applications are authored with respect to the schema elements; thus, an update in the structure might lead a query to be syntactically invalid. Similarly, even a small addition might add schema structures that a query would ideally like to take into consideration to provide a full account of the stored data, leading in semantic inconsistency. Thus, the immediate impact of schema evolution is that the surrounding source code of an information system needs to be maintained in order to be consistent with the new schema.

Developers are thus required to go to great pains in order to guarantee this consistency. The source code has to evolve also, in accordance with the new schema structure, a task that we call *schema and source-code co-evolution*. The evidence for this problem is only anecdotal (and certainly inversely related to the attention we, as a scientific community, have paid to address it). To the best of our knowledge, we can refer to only two anecdotal references on the problem:

- In [30], Stonebraker at al., note: "In a survey of 20 database administrators (DBAs) at three large companies in the Boston area, we found that . . . , DBAs try very hard not to change the schema when business conditions change, preferring to "make things work" without schema changes. If they must change the schema, they work directly from the relational tables in place. "
- In [15] , Limoncelli notes: "When the software is tightly coupled to the database schema it becomes impossible to perform software upgrades that require a database schema change. If you first change the schema,the instances will all die or at least get confused by the change; . . . Why not upgrade the instances first? Sadly, as you upgrade the instances' software one by one, the newly upgraded instances fail to start as they detect the wrong schema. You

*Work done with Univ. Ioannina.
†Work done with Univ. Ioannina.

will end up with downtime until the schema is changed to match the software"

Robert Martin, in his seminal discourse of the SOLID principles of software architecture [19], introduces *rigidity* as "the tendency for software to be difficult to change, even in simple ways; every change causes a cascade of subsequent changes in dependent modules" and *fragility* as "the tendency of the software to break in many places every time it is changed". In our previous research [36], [35], [34] we have referred to the severe impact that schema evolution has to the semantic, syntactic and operational correctness of the software applications built on top of it as *gravitation to rigidity*. Gravitation to rigidity is the main reason that makes developers and designers hesitant to evolve the schemata of the underlying databases in their information systems. Thus, there are several related questions that act as drivers for this paper:

- How extensive is the co-evolution of the schema and source code of data-intensive information system? Do schemata and source code evolve hand-in-hand, or do they evolve with different heartbeats?
- Is the conjecture of gravitation to rigidity actually supported by evidence that demonstrates the anecdotal reluctance to evolve the schema after the original releases?

Previous research on how the source and schema co-evolution takes place does exist, albeit scarce [16], [24], [10]. The absence of publicly available schema histories that predated the existence of Free Open-Source Software has been the main reason for this scarcity. However, even after the proliferation of FOSS systems, there has not been any study of substantial size or impact to provide insights in the problem. To this end, in this paper, we have embarked in an attack to the problem.

Our starting point has been the publicly available schema histories of [33]. These histories refer to 195 schemata of FOSS systems, mined from Github. In this paper, for all these systems, we have locally cloned the history of the entire system (not just the schema, as in [33]) and extracted a summary of the evolution of its source code. Then, we have studied the extent to which the source code of the system and the schema co-evolve in a synchronous way.

As one can observe in Fig. 1, depicting graphically co-evolution in a joint progress diagram, a synchronous co-evolution is observable in some cases, but not in others. *To what extent, then, is synchronicity of schema and source code co-evolution present in our 195 histories?* We introduce a normalized measure of synchronicity to measure the percentage of time-points in the history of the project where the source code and the schema demonstrate a hand-in-hand co-evolution. To define synchronicity realistically, we allow for a distance threshold which allows the schema and the source code cumulative progress of evolutionary activity to be in distance lower or equal to a factor θ (which in our measurements we have set to 10% –i.e., if at a certain timepoint the schema and source code change progression differ less than 10%, we say that they are synchronous for this time-point). *Our findings reveal that there is no single dominant behavior for the different projects, and the extent of co-evolution is uniformly distributed to all kinds of behaviors.* This is not necessarily good news: it means that the cases where co-evolution is strongly present and schemata evolve hand-in-hand with the rest of the code are rather few. Moreover, *a collateral finding reveals that after the 5^{th} year of existence, we can clearly observe a gravitation towards lower, mid-range values of synchronous co-evolution, which practically means that, with time, the schema stops evolving as actively as it did.*

Based on this finding, we moved up to investigate this property more. The next question that occupied us was *the quantification of the progress of schema change contrasted to the progress of the surrounding code's change, as well as, how early did schemata change in terms of time.* To this end, we measured how often does the cumulative progress of schema evolution exceed the respective progress of source change, as well as the respective progress of time. The results indicate that *a large majority of schemata demonstrates early advance of schema change with respect to code evolution, and, an even larger majority is also demonstrating an advance of schema evolution with respect to time, too.*

A third contribution of this paper is the study of maturity attainment and an answer to the question: *at which time point in their lives do schemata attain a substantial percentage of their evolution?* To address the question, we measured the time point in the history of the project (as a percentage of the lifetime of the project) when the projects attained a certain level of progress of their evolutionary activity. This allows to check for gravitation to rigidity: for example, if 75% of the total change is attained within the first 20% of the life of project, this is a clear indication that after an initial boost, the schema is no longer actively evolving. Our findings indicate that *almost half the projects reach 80% of their schema evolution within the first 20% of time* (which we treat as a special, schema-evolution-specific case of the Pareto principle of the vital few). At the same time, by exploiting the taxonomy of projects proposed in [33], we also observe that the increase of activity in a taxon decreases the chances of the 80/20 rule to hold; thus, we can confirm that *despite a strong tendency in a large part of the population to gravitate to rigidity for their schema evolution, there do exist projects that resist this gravitation and actively evolve the schema throughout their entire life.*

In a nutshell, the contribution of this paper is as follows:

- Our measurements replace anecdotal rumor with concrete numerical evidence for the bias of the development community towards fixing the schema early, in the general case, and the possibility of deviating from this rule of thumb in actively maintained projects.
- The paper contributes methodologically with a collection and measurement method for objectively assessing schema evolution with respect to the evolution of the surrounding code.
- The paper concludes with a discussion of the implications that these findings have for the way databases are accessed by surrounding applications and possibilities to alleviate the observed rigidity problems.

Roadmap. This paper is structured as follows. Section 2 presents the state of the art and summarizes the background, as well as the data set used, for the paper. Section 3 presents the foundational nomenclature, the research setup and the introduced metrics for schema and source co-evolution. Sections 4, 5, 6, and 7 addresses our fundamental research questions and their answers. Section 8 discusses threats to validity and Section 9 summarizes the contribution of the paper and why this matters, while also offering possibilities for future work.

2 BACKGROUND AND RELATED WORK

2.1 Works on schema evolution in general

Studies of how schemata evolve include mostly papers from the second half of the last decade [28], [5],[16], [37] (partially

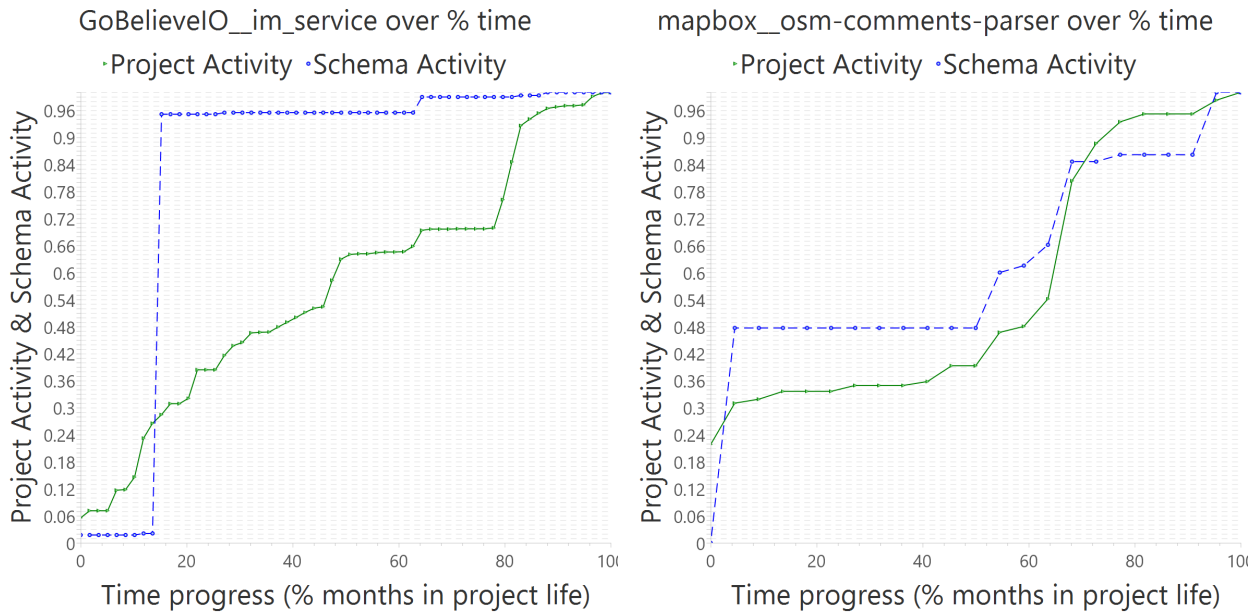

Figure 1: Examples of schema and source code co-evolution. The horizontal axis demonstrates the progress in time, as a percentage of a project's life. The vertical axis depicts the cumulative progress as a percentage of the total amount of evolution activity, for (a) the schema (dotted line) and (b) the source code (solid line).

confirmed also by [1]), [24], [3], [29], [36], [35], [7], [34], [8]. Later studies have also moved towards the study of schema evolution in the realm of JSON, NoSQL databases [14], [26] – see [31] for an overview. The handling of adaptation to schema changes includes several works [20], [9], [23], [11], [17] – see [2] for an overview of query adaptation techniques. The introduction of algebras of schema evolution operations (SMO's), in order to be able to describe sequence of changes (either in forward- or reverse-engineering) includes works like [4], [13], [27].

2.2 Works on schema and source-code co-evolution

The first study of schema evolution that we are aware of, is [28]. The author monitored a single database schema, serving a hospital application, and along with quantifying the changes in the schema, he also measured their impact to the code. A table addition resulted in an average of 19 changes in the surrounding source code, whereas a table deletion produced 59.5 changes in the application code. Attribute additions and deletions affected the source code with two and 3.25 changes, respectively.

In [16], the authors address the issue of 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 studying collateral evolution, the first contribution of the paper comes with a study of two open-source projects, Mozilla and Monotone. The findings were that the database schema and source code does not always evolve in sync, even in the presence of provisions to address the problem. To avoid conflicts with database and source code, Mozilla uses two methods, (a) 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, and, (b) 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. As a side study, the authors investigate the problem of the evolution of data format in three major database management systems, SQLite, MySQL and PostgreSQL.

In [37], the authors propose 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 of four popular applications containing embedded databases, over large time periods. The key findings of their study are (a) embedded databases are more prone to restructuring, rather than continuous growth and (b) the early stages in schemas of embedded databases tend to have a higher number of changes, while the later versions include few changes and the schema stabilizes over time.

In [24], an empirical analysis for the co-evolution of schema and code in database-based applications is made. The authors used ten popular open-source projects for their study. The authors not only study how the schema evolved on its own, but placed emphasis on the extent to which the schema and the surrounding source code were in sync. The authors claim that database schemas evolve at a high rate during their lifecycle, on average 90 atomic schema changes per year. Also, change is local in terms of location and time. In terms of location, change is located in few tables: 60%-90% of changes refer to 20% of the tables and nearly 40% of schema tables did not change. In terms of time, in 7of the 10 studied projects, their schema size approaches 60% of their maximum value within the first 20% of their lifetimes. To study applications and databases co-evolution, the authors have randomly sampled 10% of the valid database revisions and manually analyzed co-evolution. A first result showed that only half of the software changes accompanied the schema change in the same revision and only 16% of the cases showed an adaptation of the code in prior or subsequent versions too. Thus, the authors confirm the non-synchronicity of the schema and

code change reported in previous literature. Second, the authors found that the impact of a schema change to the source code is potentially important and depends on the type of change. The impact is assessed as each atomic change at the schema level is estimated at 10 - 100 lines of application code been updated per atomic schema change and 100 - 1000 lines of application code per database revision.

In [10], the authors study Oscar, an electronic medical records open-source system from the viewpoint of co-evolution of the database schema and the source application, as well as from the viewpoint of studying developer effort. The authors demonstrate that both the application and the schema grow linearly. The schema grows at a fairly constant pace; however, the growth rate is significantly lower than the one of the application, with the divergence growing steadily over time. Another interesting finding was that although pure SQL was mainly used for database access throughout the entire studied period, after a 6-year period of working with Hibernate, the developers replaced it with JPA (still, SQL remains around 80% of the database related points in the code). Developer effort is distributed in both database related and unrelated files.

In a demo paper [25], the authors approach the problem of co-evolution with a form of patching for the change that is applied to the schema that dictates (a) how the schema should be modified, and, (b) how the queries around the schema should be adapted. The patching language proposed is tailored in such a way that the SQL dialects of different DBMS vendors can be gracefully handled simultaneously for a single change.

Techniques to handle the impact of schema evolution vary. The topic is outside the scope of the paper, which is to perform a large-scale study for the extent of co-evolution in FoSS systems. We refer the interested reader to [17] for a survey of such techniques.

Finally, in [21] the authors explore the differences between biological and software ecosystems. The chapter starts with a survey of fundamental concepts in software and natural ecosystems, with examples and an emphasis on sociotechnical aspects. Then, the authors discuss fundamental concepts and theories of natural evolution and put biological and software evolution in counterpoint. Finally, the authors assess whether analogies between software and natural ecosystems do exist via the case study of a FOSS project, GNOME. The results indicate that there is no analogy, but rather different behaviors, mostly attributed to the inapplicability of the mapping of developers to antagonizing species (as, in FOSS, antagonism is not really an issue).

In [33] the author reports on his findings from a very large study of schema evolution histories. The author went on to collect the histories via Google BigQuery which has the GitHub Activity data set of 2.8 M projects and ended up with 327 histories out of which (a) 132 (40%) had just a single commit and never changed anything in the schema, and (b) 195 histories with at least an extra commit, which were subsequently analyzed. For each project, the processing produced (a) the history of the schema (which is a list of versions of the schema DDL file), (b) the automatically produced time series of changes (which is called the *heartbeat* of the schema) by pairwise comparing subsequent versions, and (c) detailed and aggregate measures of the schema history in terms of timing, schema size, numbers of tables and attributes changed, and several measures of the evolution activity of the schema.

The main contribution of the paper was the introduction of schema evolution archetypes, called "taxa", by manually clustering schemata in groups of similar evolution behavior, which are called taxa. Specifically, the taxa of schema evolution are:

(1) completely *frozen* schema histories with zero change at the logical level;
(2) *almost frozen* histories of very small change, typically with few intra-table attribute modifications;
(3) almost frozen histories but with a single spike of change and almost no other change (*Focused Shot and Frozen*);
(4) histories of *moderate* evolution, without spectacular changes, but rather small deltas spread throughout the life of a project;
(5) projects with evolution similar to the moderate one but also with a pair of spikes on their activity (*Focused Shot and Low*);
(6) histories of *active* projects, typically with a high volume of change both as intra-table change and in terms of table generation and eviction.

Based on the above taxonomy, and the distribution of projects in its taxa (which is overwhelmingly towards more frozen histories than active) it is clear that although evolution exists, its absence is much more widespread: as already mentioned, out of the largest possible collection of 327 projects that we came up, 40% had no evolution whatsoever, 10% had different versions but no schema changes at the logical level and 20% were almost frozen.

2.3 Comparison to related work

Compared to the related work, this paper is the first study to attempt a massive, large-scale study of schema and source co-evolution, by at least an order of magnitude to previous studies. The variability of the data set of [33] that is used, in terms of the different " characters" of the schemata studied adds to the validity and generalizability of the findings. The proposed measures of co-evolution and the respective visualization methods are also contributions of the paper.

3 RESEARCH SETUP

In this section, we introduce the necessary nomenclature, as well as the extraction and computation process for the datasets and measures we employ in this paper.

3.1 Extraction of Schema and Project Monthly Heartbeats

The Schema_Evo_2019[1] data set of [33] was readily available at the beginning of the effort. In a nutshell, the data set contains 195 schema histories, extracted via a meticulous and principled method, intentionally aimed towards avoiding any bias in project selection, and thus providing a representative enough collection of schema histories. The selection process for the schemata that ultimately made into the data set includes three phases:

(1) *Collection of candidate repositories from Google Cloud BigQuery.* The phase included (a) selecting .sql files and their metadata from the GitHub Activity Dataset; (b) keeping only original repositories, with more than 0 stars and more than 1 contributor from corpus that was also part of the the Library-io data set.
(2) *Elicitation of candidate repositories.* The phase included (a) the retention of single-file schema-DDL projects (either directly at the query result or after manual processing); the filtering out of projects with the terms 'example, demo, test, migrat' in their path, and (c) the choice of MySQL

[1] Available at https://bit.ly/3nMggEx (at Github) i.e., https://github.com/DAINTINESS-Group/Schema_Evolution_Datasets/tree/master/SchemaEvolutionDatasets2020

or Postgres (in that order) in the case of more than one supported vendor.

(3) *Post-processing*. This phase included the filtering out of projects with 0 or 1 versions or projects with no CREATE TABLE statements in their .sql files.

For each history of the 195 projects of the data set, several statistics are computed. Specifically, among other measures, the data set comes with measurements (in attributes) of: attributes born with a new table, attributes injected into an existing table, attributes deleted with a removed table, attributes ejected from a surviving table, attributes having a changed data type, or a participation in a changed primary key. In [33], the sum of all these updates was named *Total Activity*, or simply *Activity* and it will be the central measure that we will use to trace the amount of evolution the schema undergoes. Given these detailed measurements, and the fact that we know the commit dates for the DDL file of each project, for the purpose of the current paper, we were also able to produce the aggregate measurements per month of the evolution activity of each schema, to which, in this paper, we will refer as *Monthly Schema Activity*, or simply *Schema Activity* (as the time-unit of all measurements is no more the individual commit, but rather the month). The *Schema (Monthly) Heartbeat* is the linear sequence of the Monthly Schema Activity measurements (with zero activity for the months without updates).

As the goal of the paper is to contrast the evolution of the schemata to the evolution of the projects that encompass them, we had to find a way to assess how much the source code of the entire project changed, and, at the same time, match it, in a compatible way, against the schema activity. To this end, we locally cloned all the involved projects, and, assessed the number of files updated in each commit (via a " git log –name-status –no-merges –date=iso" command) that produces, for each commit, the names of the changed files, the date, and some extra information on the authors and their messages. Then, via custom scripts, we measured the number of updated files per commit, and, knowing the commit dates, we were able to produce the total number of updated files per month, which, in this paper, will be referred to as *Monthly Project Activity*, or simply *Project Activity*. The *Project (Monthly) Heartbeat* is the linear sequence of the monthly Project Activity measurements (with zero schema activity for the months without updates). In all of these tracked changes that concern the evolution of the project, we have subtracted the changes of the DDL file, to clearly separate schema and source evolution.

A couple of parenthetical notes is due here:

- Moving from the individual commit to the month as the time-unit is (a) necessary for establishing a fixed progression step in the time axis, (b) convenient for comparing schema and project activity over the same time points, and, (c) a fair compromise between more detailed granules of time (like day) and coarser units (like years in the time axis, or version releases, for which no information is automatically attainable).
- How representative is the measure we use for the activity of the project? Certainly, assessing the amount of change requires a unit of change. We avoid detailed measures such as lines of code and use a coarser unit, such as the files changed. We understand that this induces the risk of equalizing changes like a single-line comment versus the creation of a new business logic class. However, we also believe that despite the obvious approximation that our method uses, the number of changed files as an indicative measure of the heartbeat of the project activity is fairly common: for example, this is how GitHub measures code frequency (the number of commits per month would be another possible measure, if we follow the same measures with GitHub).

3.2 Cumulative Measurements Used for the Analysis

Finally, to be able to compare the schema and project heartbeats, we normalized them both via their cumulative percentages. Assume a specific heartbeat, with the amount of activity per month. We can compute the total activity that occurred in the entire lifetime of the heartbeat as the sum of all monthly activity measurements. Then, we can compute the heartbeat with the percentage of total activity per month, instead of the actual measurement, via a simple division to the total lifetime activity.

Definition. The *cumulative fractional activity of a heartbeat* is the running total of the percentage values of a heartbeat and is a monotone time-series progression of how change accumulated over time to reach the 100% of total lifetime activity. The formula for the cumulative percentage is as follows:

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

with $activity_k$ 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.

For example, if the percentage of a project's progression in 4 successive months is 40%, 25%, 20%, and 15%, respectively, the cumulative percentage values would be 40%, 65%, 85%, and 100%, for each month. The above formula obviously applies to all kinds of activity measurements. Similarly, we can apply this transformation to time and assign a percentage of time progress to each month, signifying how much ahead in time the project was with respect to its life-span.

To compare the two heartbeats, we therefore produced, for each project, the following time-series: (a) the cumulative fractional schema activity, (b) the cumulative fractional project activity, and (c) the respective cumulative fractional time progress. A *joint (cumulative fractional) progress diagram* combines these three elements into a single diagram, like the one of Figure 1.[2]

3.3 A case study

In this section, we briefly present the case study of one of the 195 projects, specifically the project `mapbox/osm-comments-parser`. The `mapbox/osm-comments-parser` project was written in JavaScript and consists of parsers to read Notes and Changeset XML files and save them in a Postgres DB. The project started in 2015 and it was active for 2 years. The owner of the repository is the Mapbox organization with 55 people and 849 repositories on GitHub.

The Project Update Period was 22 months and the Schema Update Period was close at 20 months. The project included 119 commits and 259 file updates. Concerning schema evolution, there were 13 commits for the schema file, out of which 9 were

[2]Data, code & results can be found at https://github.com/DAINTINESS-Group/Schema_Evolution_Datasets/tree/master/SchemaEvolutionDatasets2020

active, i.e., they included changes to the schema. The co-evolution of source code and schema for the project is depicted in the right-hand side of Figure 1. Due to the similar schema and project update periods, the depiction of cumulative progress with respect to the actual time is very similar to its depiction with respect to time progress (as in Fig.1). Observe that the schema starts earlier with a 48% of change at start-up, then stabilizes until 50% of the project's life and then starts to attain more changes along with the evolution of the source code. There are two flat-line periods of no change connected by a period of incremental change. The project has attained 50% of the schema changes at 55% of its life and 80% of the schema changes at 68% of its life. The cumulative progress of the schema and the source were close (i.e., with less than 10% difference) for 43% of the time.

We studied the different commits to the schema and the commits to the source code in a small window of changes before and after them to get an idea of what happens to the schema and code when the schema changes. A detailed analysis and presentation of 6 projects can be found at the accompanying website of the paper. Here, we discuss the `mapbox/osm-comments-parser` project only. Concerning what changes too place we observed the following types of change: (i) table creation and deletion, accompanied by the respective source updates, (ii) attribute addition and deletion, again with changes at the code, (iii) multiple data type changes, (iv) bug fixing related to all these changes, (v) feature additions, (vi) refactorings of the code and restructuring of the placement of the files in folders. Concerning *who* did the changes, 90% of the studied updates were performed by the same developer. Concerning *when* changes happened, observe how the changes are distributed over time at the beginning and the second part of the project's life. Concerning *where* the changes happened in the source code, updates took place in a group of files related to the database.

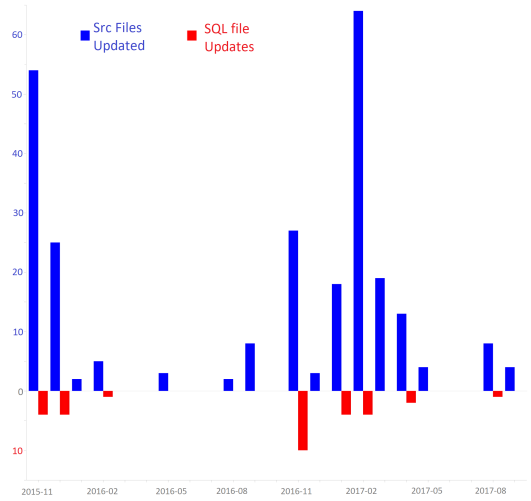

Figure 2: Monthly heartbeat of the `mapbox/osm-comments-parser` project (up, blue) and schema (down, red)

3.4 Research Questions

Having described how all the data have been computed and available, we can now proceed with the core of this paper's contribution.

We start with the fundamental research questions that drive our research:

- **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**: Is it legitimate to say that schema evolution precedes source code evolution? Do schema changes come early in time, and/or earlier than source changes?
- **Research Question 3**: At which point in the project's lifespan do schemata complete a substantial part of their evolution?

In the following sections, we discuss our approach towards addressing these questions and present our findings.

4 IS SCHEMA EVOLUTION IN SYNC WITH SOURCE CODE EVOLUTION?

The first research question aims to measure the extent to which the schema evolution follows the projects' evolution.

[RQ1] What percentage of the projects demonstrates a synchronous, "hand-in-hand" schema and source code co-evolution?

To assess the extent of co-evolution, we exploit that time has been quantized into months, and we provide a measure of how many times in the lifetime of the project the cumulative progress of the schema was " far-away" from the respective cumulative evolution of the entire project.

Formally, assume two heartbeat sequences P=$[p_0, \ldots, p_n]$ and S=$[s_0, \ldots, s_n]$ over a time period of n time-points T=$[t_0, \ldots, t_n]$. The 0-th elements refer to the initiating commit with the creation of the respective construct (e.g., DDL file or entire project). We introduce two measures of synchronicity.

Definition. For a specific timepoint t_i, we say that the predicate $\theta - synchronous(t_i)$ is true if $|p_i - s_i| \leq \theta$, with θ being an arbitrarily set threshold.

Definition. The θ−$synchronicity$ of P and S, θ−$synchronous(P, S)$, is the fraction of the time-points that are $\theta - synchronous$ over the total amount of n points.

The intuition of the measure in our case is that we count how many times in the lifetime of the project the cumulative advance of activity for the two heartbeats was close enough, within an arbitrarily set band of θ. In our case, we perform the comparison of the two heartbeats of (a) cumulative fractional project activity vs (b) cumulative fractional schema activity and measure which percentage of time the two heartbeats were in the respective synchronicity. We also fix the value of θ, which has to be relatively small, in order to characterize a reasonable difference for a " hand-in-hand" co-evolution between the two heartbeats. Remember that θ is not a measure of lag, but just an acceptance band for " hand-in-hand" co-evolution; it is the θ-synchronicity that will measure the percentage of time where a lag was present. We have performed the comparisons for two values of θ, 5% and 10%; we report the results of 10%-synchronicity only (their Kendall correlation is 0.67).

In Figure 3, we present 6 projects as examples, each from a different taxon, where the schema and source code co-evolution are synchronous in a large percentage of the projects' life for the first three of them and out of sync for the last 3 ones.

As the figure demonstrates, the behaviors of the different projects with respect to the synchronicity of the co-evolution are different. In order to be able to understand a global picture of the

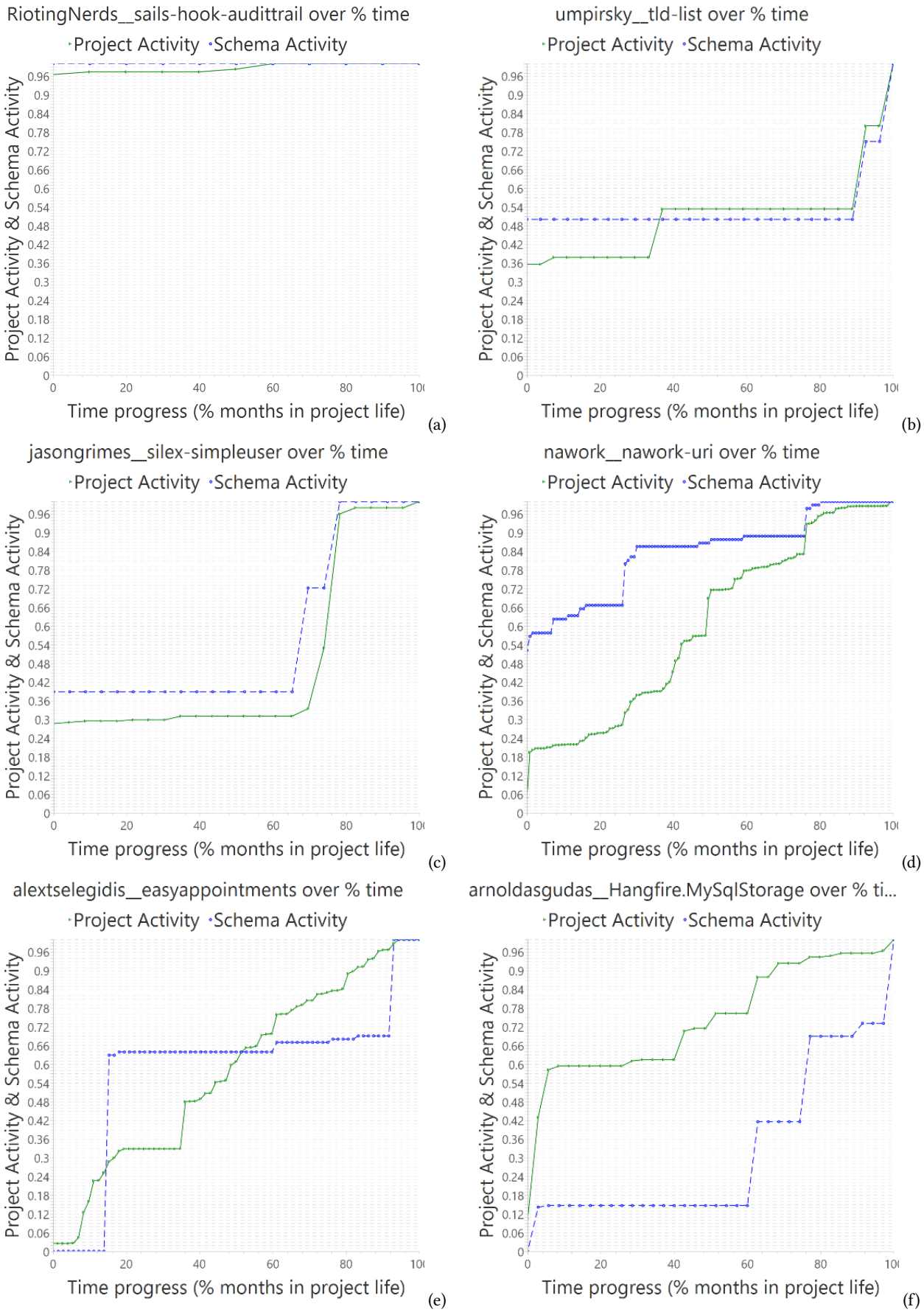

Figure 3: Line charts with synchronous co-evolution for the taxa: (a) FROZEN, (b) ALMOST FROZEN, (c) FocusedShot n FROZEN; and out-of-sync co-evolution for the taxa (d) MODERATE, (e) FocusedShot n LOW, (f) ACTIVE.

behavior of the data set overall, we had to resort in grouping the projects in histograms of their θ-synchronous measure. To this end, we grouped the projects into five buckets ([0%-20%) - [20%-40%) - [40%-60%) - [60%-80%) - [80%-100%]), with each bucket covering a range of the θ-measure value. Then, each project is allocated to the respective bucket - for example, if a certain project has a θ-synchronous value of 55% (i.e., for the 55% of the months in its lifetime, the two cumulative fractional heartbeats had an absolute difference lower or equal than θ), then this project is allocated to the 40%-59% bucket. The distribution of projects to histogram buckets for the entire data set is shown in Figure 4.

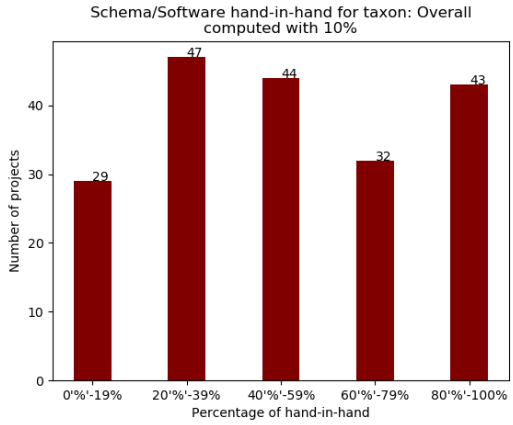

Figure 4: Breakdown of projects per value range for the 10%-synchronous co-evolution

Findings. The main message based on what we observe in the figures is that *there are all kinds of behaviors with respect to project and schema co-evolution, both overall and within in the different taxa.*

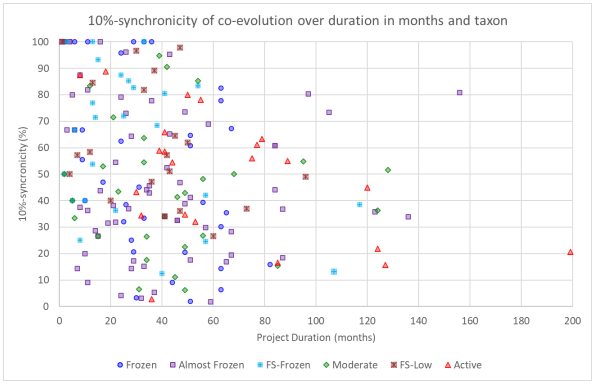

Figure 5: Correlation of duration and co-evolution synchronicity per taxon

In Figure 5, we depict the scatter-plot of all projects, separated by taxon with respect to their duration and 10%-synchronicity of co-evolution. In the figure, we can observe that there is a box, of durations up to 60 months where all behaviors are present (i.e., synchronicities of up to 100%). However, as usually happens in this line of research, empty spaces in the chart " speak" too: see the empty spaces in the band between 60 and 140 months. *After exceeding this 5-year threshold of 60 months, there is a gravitation towards lower, mid-range values of synchronous co-evolution, which practically means that, with time, the schema stops evolving as actively as it originally did.*

5 HOW PREMATURE IS SCHEMA EVOLUTION COMPLETION?

As we already saw in the previous section, the cumulative synchronization numbers reveal that schema evolution is not uniformly spread over time; moreover, schema evolution seems to get out of sync with project evolution after a certain period of time. A study of the individual joint progress co-evolution diagrams clearly verifies this property.

In our previous research [36], we have observed, in various forms and patterns, a phenomenon which we named *gravitation to rigidity*: "in the realm of schema evolution, rigidity is a stronger force than change, as we see that databases have a tendency to avoid change and evolution in favor of calm lives, thus making them rigidity-prone rather than evolution-prone". Previous research has made claims corroborating the gravitation to rigidity: [24] reports that in 7 of the 10 studies projects, 60% of the schema changes are completed in the first 20% of the projects' life.

All this background gave us the motive to exploit our collected histories and measurements and investigate completion rates: to the extent that schema evolution activity is not uniformly spread in time, *is it legitimate to claim that schemata collect their evolution quickly after birth, and live calmer lives later?*

Given the cumulative progression of change in the schema and the project heartbeat, we measure how often is the schema progression in advance of (a) the respective project progression, and (b) time.

At any particular time point (in our case: month), the measure of "advance" of a heartbeat over another is a higher cumulative progress of activity. Then, the research question that we ask is *[RQ2] how probable is it that a project will have schema evolution preceding source evolution, and, in particular, for what percentage of its life?*

Intuitively, if at the end of a certain month, the cumulative progression of time is 40%, the cumulative progression of the schema evolution activity is 50% and the cumulative progression of the source code evolution activity is 55% we can say that, for this month, the schema is *in advance* of time, with respect to its cumulative progress (i.e., it has collected more change than if change evolved at a steady pace) and (b) the schema is *lagging* or *late* with respect to the source code (i.e., the collective rate of the source code, so far, exceeds the one of the schema). Of course, the result of the comparison of schema progress to the other two measures is different per month – it is only the last month where all cumulative heartbeats end up in 100%.

5.1 For what percentage of a project's life is schema progress in advance of source progress or time?

To this end, we have counted the number of times that the schema is not lagging with respect to time and source, and to make the numbers normalized, we have divided them by the months after the creation of the project. Practically, we get the percentage of the project's life (in months) where the schema is not behind the time and the source progress.

Definition. We define as the *life percentage of schema advance over time (resp., source)* the fraction of (a) the number of months where the difference of the cumulative fractional activity of the

schema minus the cumulative fractional progress of the time (resp. source) was larger or equal to zero, over, (b) the months of the project's life after its creation.

Practically, we measure, (a) for how many months, and, (b) straightforwardly, for which percentage of the project's life, has the schema cumulatively progressed more, in terms of its evolution, over time or source code. In Figure 6, we depict the breakdown of projects in ranges of life percentage of schema advance over (a) source, and, (b) time. For each range, we report the number of projects that pertain to this range, its percentage over the total population of 195 projects and its cumulative percentage starting from the higher valued ranges.

% project life where schema is in advance of ...

	...Source			...Time		
Range	#prjs	%	Cum. %	#prjs	%	Cum. %
0.9-1.0	79	**41%**	**41%**	100	**51%**	**51%**
0.8-0.9	20	**10%**	**51%**	21	**11%**	**62%**
0.7-0.8	13	7%	57%	7	4%	66%
0.6-0.7	9	5%	62%	10	5%	71%
0.5-0.6	17	9%	**71%**	14	7%	**78%**
0.4-0.5	11	6%	76%	9	5%	83%
0.3-0.4	10	5%	82%	8	4%	87%
0.2-0.3	9	5%	86%	10	5%	92%
0.1-0.2	11	6%	92%	10	5%	97%
0.0-0.1	14	7%	99%	4	2%	99%
(blank)	2	1%	100%	2	1%	100%
Grand Total	*195*	*100%*		*195*	*100%*	

Figure 6: Life percentage of schema advance over time and source code

The main findings revealed by the numbers of Figure 6 are:

- *The large majority of projects, demonstrate an advance of schema evolution progression over the progression of source code evolution or time.* Specifically, 41% of the projects had at least 90% of the months of their lives with the cumulative schema evolution progress being higher than the respective source code cumulative evolution. *Concerning time*, things are even more impressive: *more than half of the projects had their cumulative schema evolution progress being higher than time progress for at least 90% of their lives.*
- Concerning the cases where schema was mostly in advance of source code or time, we observe that *71% of the projects had schema evolution in advance of source code evolution for at least 50% of the project life, and 78% of the projects for time, respectively.*

We believe this is a clear demonstration that developers prefer contributing changes to the schema structure early in the project life and this makes schema change being cumulatively in advance of time and the rest of the source code. We should also note that the two measures of advance over (a) time and (b) source code are very highly correlated, with a Kendall correlation of 0.75.

5.2 For what percentage of projects is schema progress always in advance of source progress or time?

The high concentration of values in the 0.9 - 1.0 category hides the fact that there are several projects with exactly 1.0 for the values of life percentage of schema advance over time and life percentage of schema advance over source. Even more interesting is the fact that it is possible that both measures are 1.0, which means that a project, for all months, demonstrates an advance of schema evolution over both time and source.

Our measurements indicate that there is a large number of projects where the progression of schema evolution is in advance of source evolution, time, or both for the entire life of the project. Specifically, the numbers suggest that schema is always in advance of (a) time, for 80 projects (i.e. 41%), (b) source code, for 57 projects (i.e., 29%) and, (c) both of them, for 55 projects (i.e., 28%) of projects. The numbers are really high, if one considers that we investigate a total dominance of schema evolution, for all months, as well as the existence of several projects where the DDL file appeared later in the life of a project (making them non-eligible for an "always" precedence of schema evolution). Clearly, case (c) should be a subset of the two others – however, observe how close cases (b) and (c) are. This implies that it is very unlikely that a schema advance over the source is not accompanied from an advance over time (but not vice versa). In this case, we can safely guess that both source and schema evolve early.

Even more intriguing was the study of the drill-down of these numbers with respect to the different taxa. Figure 7 shows how the different taxa contain projects where schema is in advance of both time and source. *A clear message is that the more frozen a taxon is, the higher its probability to demonstrate an early advance of schema over both time and source code. The resulting consequence for these cases is that we have as clear evidence as possible for the essence of gravitation to rigidity: perform a few changes to the schema -if any- early and then stay rigid.*

6 HOW EARLY DOES SCHEMA EVOLUTION ACTIVITY OBTAIN A CRITICAL MASS?

Apart from answering the "how often" precedence question, it is also interesting to find out how early in time does the activity of changing the schema reach a substantial fraction of its entire volume.

6.1 Alpha-attainment fractional timepoints

In an attempt to summarize in a single number the prematureness of schema evolution over time, we embarked on studying at which time point in the live of a project's history do schemata attain certain levels of completion. To give a specific example, given a specific schema, the question one might want to ask is "at which time point in the project's life did the schema attain 75% of its evolution?", with 75% being an arbitrarily set level of evolution completion. Concretely speaking, we need to make the following definitions of evolution completion.

Definition. Assuming a specific schema, and its cumulative fractional activity of schema evolution, we define as the *α-attainment timepoint* the timepoint at which the cumulative fractional activity reaches or exceeds an arbitrarily-specified threshold α, with α signifying a percentage of the total schema evolution activity.

For example, assuming that the cumulative fractional activity of schema evolution is [20%, 47%, 85%, 95%, 100%, 100%, 100%] for months M0 (creation of the schema) to month M6 (the last month of activity of the project), the 45%-attainment timepoint is the month where the schema evolution activity reaches or exceeds 45%, i.e., month M1.

Definition. We define as the *α-attainment fractional timepoint* the percentage of the project's life covered by the α-attainment

Time lags schema always	FROZEN (34 prjs)	ALMOST FROZEN (65 prjs)	FocusedShot n FROZEN (25 prjs)	MODERATE (29 prjs)	FocusedShot n LOW (20 prjs)	ACTIVE (22 prjs)	*Grand Total (195 prjs)*
FALSE	50.00%	47.69%	64.00%	62.07%	80.00%	68.18%	*57.95%*
TRUE	50.00%	50.77%	36.00%	37.93%	15.00%	31.82%	*41.03%*
(blank)	0.00%	1.54%	0.00%	0.00%	5.00%	0.00%	*1.03%*
Grand Total	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*

Source lags schema always	FROZEN (34 prjs)	ALMOST FROZEN (65 prjs)	FocusedShot n FROZEN (25 prjs)	MODERATE (29 prjs)	FocusedShot n LOW (20 prjs)	ACTIVE (22 prjs)	*Grand Total (195 prjs)*
FALSE	52.94%	64.62%	72.00%	72.41%	90.00%	86.36%	*69.74%*
TRUE	47.06%	33.85%	28.00%	27.59%	5.00%	13.64%	*29.23%*
(blank)	0.00%	1.54%	0.00%	0.00%	5.00%	0.00%	*1.03%*
Grand Total	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*

Both time and source lag schema always	FROZEN (34 prjs)	ALMOST FROZEN (65 prjs)	FocusedShot n FROZEN (25 prjs)	MODERATE (29 prjs)	FocusedShot n LOW (20 prjs)	ACTIVE (22 prjs)	*Grand Total (195 prjs)*
FALSE	52.94%	66.15%	76.00%	72.41%	90.00%	86.36%	*70.77%*
TRUE	47.06%	32.31%	24.00%	27.59%	5.00%	13.64%	*28.21%*
(blank)	0.00%	1.54%	0.00%	0.00%	5.00%	0.00%	*1.03%*
Grand Total	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*

Figure 7: Percentage of projects where schema is advance over time or source code for the entire life of the project

timepoint (thus measuring time as a percentage of the project's lifetime instead of measuring the actual month).

In the previous example, with a schema reaching 45%-attainment in month M1, in a duration of 6 months, the α-attainment fractional timepoint is 1/6, i.e., 16.66%.

In a fully linear scenario, if schema evolution was performed with a constant rate, a certain percentage of the schema evolution would have been obtained at an equal percentage α of time (i.e., 45% of the total evolution activity would have taken 45% of time to occur). All our -so far, mostly anecdotal- evidence was that this was not the case. The research question that is formed then is:*[RQ3] at which time point in their lives do schemata attain a substantial percentage of their evolution?*

6.2 Attainment of a large part of evolution activity over time

To address the question, for each project, we measured several attainment (fractional) timepoints, for different completion rates, and specifically, 50%, 75%, 80% and 100%. In Figure 8, we can see the overall breakdown of projects (i.e., independently of taxa) with respect to when they reached a specific level of completion schema activity. The bar chart is to be interpreted as follows:

- The horizontal axis refers to the percentage of schema activity measured, i.e., the α threshold.
- 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 (a) what percentage of schema activity was completed within (b) this range of the time.

Take for example the second quadruple of bars in the figure. The horizontal axis says that we measure when 75% of the projects' evolution activity was attained. The 4 series tell us that: (a) 98

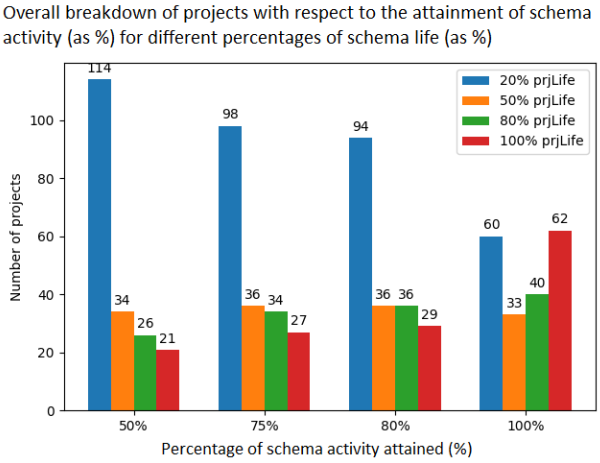

Figure 8: Breakdown of projects with respect to when (as a percentage of their lifetime) they reached a specific level of completion schema activity

projects had attained 75% of evolution activity by the time they reached the first 20% of their lifetime; (b) 36 projects attained the 75% of their evolution activity between 20% and 50% of their lifetime; (c) 34 projects between 50% and 80% of their lifetime, and (d) 27 projects after passing 80% of their lifetime.

Understandably, the earlier projects reach a substantial attainment level, the more premature their completion is, and, *when the completion is attained very early, the stronger the gravitation to rigidity is.* Thus, seeing that, 98 of the 195 projects (i.e., 50%) attained 75% of the evolution in just the first 20% of their project's lifetime is a very strong indication of gravitation to rigidity. On the other hand, *resistance to rigidity also exists*: for 27 projects

(14%), the completion was late, as they had to exceed the 80% of their lifetime to attain a 75% of evolution.

To present the findings, we will discuss the third group of measurements that pertain to an α threshold of 80% of the evolutionary activity. We chose 80% as it is related to the well-known 80/20 rule [22], also known as the Pareto principle which states that for many outcomes, roughly 80% of consequences come from a " vital few" 20% of causes. 94 of the 195 projects (48%) attained an 80% completion within the first 20% of their lifetimes. Here, the 80/20 rule holds for half the projects of the collection. Adding to these projects the 36 projects of the (20%-50%] interval, we see that 130 of the 195 projects, i.e., exactly the 2/3 of the data set, had reached 80% of their evolution in the first half of their life. Another 36 projects (18%) were required to bring this to a "linear" equilibrium: 80% of the evolution in 80% of the time. Only 29 (15%) of the projects had a late completion rate, beyond 80%.

The rightmost group of attainment of 100% of evolution, gives even stronger evidence for gravitation to rigidity:

- 60 out of 195 projects (31%) had reached 100% of their schema evolution activity in the first 20% of their life
- 93 projects (48%) had reached 100% of schema evolution in 50% of the time

On the other hand, another 31% of the projects (62) reached their 100% completion after 80% of their project lifetimes. This means that there is a part of the population that resists rigidity and demonstrates changes also at late timepoints.

Overall, we can argue that a very large amount of projects, attract schema changes early in their project life, with 1/3 of the projects having attained the entire evolution in the first 20% of their project life and 1/2 of the projects attaining 100% of schema evolution in the first half of their life. At the same time, there is also a remarkable part of the projects that resist this gravitation to rigidity, and, display schema evolution throughout a large part of their lifetime.

7 STATISTICAL ANALYSIS

Normality tests. All Shapiro-Wilk tests of normal distribution, for all attributes, produced p-values lower than 0.007 (i.e., there is no normality in the distribution of any attribute).

Testing synchronicity. We tested taxon over 10% synchronicity via a Kruskal-Wallis test, with 0.05 as alpha level. The p-value was 0.003, an order of magnitude less. The taxon has a moderate effect. The two focused shot taxa have the highest medians with respect to 10% synchronicity, with 68% for *FocusedShotAndFrozen*, and 57% for *FocusedShotAndLow*, respectively: code evolves closely with schema in "shot-oriented" projects. The *Active* taxon has a median of 55%: actively evolving schemata are parts of projects with source evolution at higher rates. The rest of the taxa had a median between 0.43 and 0.48.

Testing attainment. We tested taxon over 75% attainment via a Kruskal-Wallis test, with 0.05 as alpha level. The p-value was 0.006, an order of magnitude less. The taxon seems to have a relatively moderate effect to the attainment. The 3 frozen taxa attain their 75% of schema change activity quite early with a median value lower than 0.2 (i.e., they attain 75% of change before the 20% of the Project Update Period). The two moderate-change taxa, Moderate and FocusedShotAndLow have a median value between 0.2 and 0.3, and the active taxon has a median value 0.47.

Testing Lag. We also performed statistical tests concerning whether the different taxa behave differently concerning the case of (i) time, (ii) source or (iii) both being found *always* in lag with respect to the schema evolution. We performed a Chi-square as well as a two-sided Fisher test for all the three categories depicted in Fig. 7, assuming an α level of 0.05. Concerning the Time Lag, both the Chi-square and the Fisher test produce a p-value of 0.07. Concerning both the Source lag and the "both-lag", the Chi-square tests give a p-value of 0.02 and the Fisher test a p-value of 0.01. Thus, concerning the issue of whether different taxa have a different behavior on how the schema evolves in advance of time, source code or both, the two last categories are statistically significant.

Other tests. We also tested via Kruskal-Wallis tests the relationship of synchronicity and attainment with other project characteristics, such as schema and project activity and duration. There is small impact of these characteristics to the studied measures; all the material is in the paper's accompanying website.

8 THREATS TO VALIDITY

To a very large extent, exactly because our data set is the one of [33], several threats to validity are common in the two papers. For the self-autonomy of the paper, we discuss the common threats here too, albeit in a more concise manner, and refer the interested reader to [33] for extensive argumentation.

Scope. The scope of the paper concerns the co-evolution of (a) the logical level of relational schemata and (b) the source code of their hosting project for meaningful Free Open-Source Software projects, hosted in GitHub. We are not covering or generalizing to proprietary schemata outside the FoSS domain. We do not cover conceptual or physical schemata. We are also restricted in relational schemata and not XML, JSON, or another format.

Internal validity. To the extent that we make no causation claims (but restrict ourselves to conjectures on such matters), we believe the paper is safe from this type of threats.

Having said that, there are several confounding factors that can affect schema evolution. Firstly, the development style of a team is greatly related to how source code and schema co-evolve. Although there is no explicit data to support this, a widely accepted conception is that the schema definition and stabilization has to be performed before serious development of database-related code begins, exactly because of the impact schema evolution has. On the other hand, more agile development styles that pay meticulous care to track where and how code depends on schema, present more degrees of freedom for schema change. Second, the effects of schema modification to runtime systems cannot be understated, as schema change as well as data migration might require a painful system shutdown for long periods (see [15]). The role of the usage of Object-Relational Mappings in the co-evolution of source code and schema is still unclear [6].

External Validity. The external validity refers to the possibility of generalizing the findings of a study to a broader context. Much as in [33], we claim that our elicited repositories and their extracted history give a fairly representative view of schema evolution in FoSS projects. exactly due the criteria used for constructing the data set corpus. Overall, (a) the thorough collection and pre-processing of the chosen projects, in a way that isolates meaningful projects, by excluding toy, demo, test, or similar schemata, as well as (b) the variety of application domains for the collected histories (which include e.g., Content Management Systems, IoT Management on the cloud, Messaging Platforms, Web on-line stores, On-line Charging Systems (OCS) and others) assert that the projects used are (i) indicative enough and (ii)

representative of how FoSS projects with relational databases evolve their schemata.

Experimental Reliability. We tested our extraction scripts and, although nobody can ever exclude the possibility of bugs, we believe in their validity.

Construct validity. We have to highlight that the most potential threat to validity of this paper is construct validity: to which extent do the metrics employed are actually representing evolution activity accurately? Our unit of change is the file and the existence of changes to a file in a certain commit. Our unit of time is the month. For the latter, we are pretty confident: we measure projects of several years' length, and thus, a month as time unit for both the schema and the project is a reasonable, common chronon. For the former, we can understand that the measure is an approximation of the total change; however, it is a quite reliable, and, importantly, accurately measured (as it is automatically extracted from git), measure. We list the construction of algorithms for a fully automated, and thus applicable at large scale, source code change detection, as well as the definition of a more precise unit of change, as opportunities for future work.

9 DISCUSSION

The question on whether schema and source code evolve hand-in-hand is important in the sense that it establishes how flexible and maintainable the database, as a component of the architecture of the information system, is. The anecdotal impression around the issue has traditionally been that the schema is the first element of the architecture to stabilize, before embarking in serious development of data-related parts of the source code. However, this anecdotal impression has never been established concretely. In this study, we give concrete evidence that there is only a 20% of the projects where evolution of schema and source go hand-in-hand and, this percentage also includes the small subset of the projects that are practically frozen too. Overall, we now have much more solid knowledge that the "build-n-freeze" mode of handling the schema of a software project –which we have also called "gravitation to rigidity" – is indeed happening. At the same time, the study also gives evidence for exceptions to this pattern: there do exist projects where the schema is actively maintained throughout the entire life of the project.

A second question concerns the comparison of the cumulative progress of schema evolution to the evolution of the surrounding source code, as well as to time progression. We have found that a very large number of schemata come with an early stabilization of their structure. When compared to source evolution, the schema also precedes the source progress – albeit to a less extent than time. Overall: schema evolution typically happens comparatively early, and earlier than source evolution.

Finally, having established that the schema is quite often a rather inflexible structure, we asked "how soon does the schema get fixed?" The statistics collected indicate a clear sign of gravitation to rigidity: for many projects, after a certain time point soon after initiation, the schema pretty much stops to evolve.

Implications. Given all the above, what are the implications of our findings? In the sequel, we present a set of implications in the form of open problems that concern (a) the research community, (b) the vendors of data-driven application development environments, and, (c) the vendors of DBMSs.

- The *research community* and *the vendors of data-driven application environments* should invest more time and effort towards the provisioning of automated tool support that enables the identification of (a) the parts of the code affected by a schema change, and, (b) the parts of the schema that require maintenance once the application code evolves, with high precision and recall. This task is extremely difficult due to the heterogeneity of the application architectures and programming languages, as well as due to the dynamic nature of queries [18, 20, 25].
- The *research community* and *the vendors of data-driven application environments* should consider providing automated tool support for refactoring, testing and migrating application code, database schemata and data. This task is also a big challenge, considering the aforementioned heterogeneity and dynamic aspects of the problem.
- The *research community* and *the vendors of DBMSs* should explore the possibility for DBMS architectures to maintain multiple versions of the schema, such that applications can bind to the version that guarantees their correctness. So far, some pioneering efforts have been made (see [12], Oracle Edition-Based Redefinition[3]). However, the issue is open for further research, while the practical implications for having "a git for schema and data" are still largely unexplored.
- Isolating the application code from from accessing the logical schema of the database, and instead, be insulated by its changes, can be done -as usual- by introducing an intermediate layer that give to each application a dedicated "adaptor" schema. This is not a new idea: such layerings have been assumed since the very beginning of relational systems [32]. However, the price to pay for this approach is the effort for maintaining the mapping integrity among the different layers. The *research community* and *the vendors of DBMS's* should provide automated facilities that reduce this effort and make the lives of the application developers and the DBAs easier.
- A departure from the relational model is progressively taking place [31]. Replacing the assumptions of our current state of practice with new ones, dictated by the very nature of self-describing data and solving all of the aforementioned issues into this new context is also an interesting challenge for the research community, the vendors of the vendors of data-driven application development environments, and the vendors of DBMSs.

Consequences and Future work. We believe that gravitation to rigidity is established after this paper. Typically, schemata evolve earlier than the code, and now we have evidence for this phenomenon, which was so far simply rumor-based. To explain the gravitation to rigidity, we have made a conjecture that the developers' reluctance to actively maintain the schema is due to the effect that schema evolution has to the surrounding code (i.e., crashes and semantic inconsistencies) and the resulting effort that is required to maintain the application code that works on top of the evolving schema. Deep verification of this aspect is still open. The construction of more precise detection methods for measuring change, that can be automated at a very large scale, is another possible path for future research.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for helping improve the paper's clarity (case study, scope and influencing factors).

[3] docs.oracle.com/database/121/ADFNS/adfns_editions.htm

REFERENCES

[1] Dimitri Braininger, Wolfgang Mauerer, and Stefanie Scherzinger. 2020. Replicability and Reproducibility of a Schema Evolution Study in Embedded Databases. In *Advances in Conceptual Modeling - ER 2020 Workshops CMAI, CMLS, CMOMM4FAIR, CoMoNoS, EmpER, Vienna, Austria, November 3-6, 2020, Proceedings (Lecture Notes in Computer Science)*, Georg Grossmann and Sudha Ram (Eds.), Vol. 12584. Springer, 210–219.

[2] Loredana Caruccio, Giuseppe Polese, and Genoveffa Tortora. 2016. Synchronization of Queries and Views Upon Schema Evolutions: A Survey. *ACM Trans. Database Syst.* 41, 2 (2016), 9:1–9:41.

[3] Anthony Cleve, Maxime Gobert, Loup Meurice, Jerome Maes, and Jens H. Weber. 2015. Understanding database schema evolution: A case study. *Sci. Comput. Program.* 97 (2015), 113–121.

[4] Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. 2013. Automating the database schema evolution process. *VLDB J.* 22, 1 (2013), 73–98.

[5] C. Curino, H. J. Moon, L. Tanca, and C. Zaniolo. 2008. Schema Evolution in Wikipedia: toward a Web Information System Benchmark. In *Proceedings of ICEIS 2008*.

[6] Alexandre Decan, Mathieu Goeminne, and Tom Mens. 2017. On the Interaction of Relational Database Access Technologies in Open Source Java Projects. *CoRR* abs/1701.00416 (2017). http://arxiv.org/abs/1701.00416

[7] Julien Delplanque, Anne Etien, Nicolas Anquetil, and Olivier Auverlot. 2018. Relational Database Schema Evolution: An Industrial Case Study. In *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 635–644.

[8] Konstantinos Dimolikas, Apostolos V. Zarras, and Panos Vassiliadis. 2020. A Study on the Effect of a Table's Involvement in Foreign Keys to its Schema Evolution. In *Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings (Lecture Notes in Computer Science)*, Gillian Dobbie, Ulrich Frank, Gerti Kappel, Stephen W. Liddle, and Heinrich C. Mayr (Eds.), Vol. 12400. Springer, 456–470.

[9] Spyridon K. Gardikiotis and Nicos Malevris. 2009. A two-folded impact analysis of schema changes on database applications. *Int. J. Autom. Comput.* 6, 2 (2009), 109–123.

[10] Mathieu Goeminne, Alexandre Decan, and Tom Mens. 2014. Co-evolving code-related and database-related changes in a data-intensive software system. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*. IEEE Computer Society, 353–357.

[11] Michael Hartung, James F. Terwilliger, and Erhard Rahm. 2011. Recent Advances in Schema and Ontology Evolution. In *Schema Matching and Mapping*, Zohra Bellahsene, Angela Bonifati, and Erhard Rahm (Eds.). Springer, 149–190.

[12] Kai Herrmann, Hannes Voigt, Torben Bach Pedersen, and Wolfgang Lehner. 2018. Multi-schema-version data management: data independence in the twenty-first century. *VLDB J.* 27, 4 (2018), 547–571. https://doi.org/10.1007/s00778-018-0508-7

[13] Kai Herrmann, Hannes Voigt, Jonas Rausch, Andreas Behrend, and Wolfgang Lehner. 2018. Robust and simple database evolution. *Inf. Syst. Frontiers* 20, 1 (2018), 45–61.

[14] Meike Klettke, Hannes Awolin, Uta Störl, Daniel Müller, and Stefanie Scherzinger. 2017. Uncovering the evolution history of data lakes. In *IEEE International Conference on Big Data, BigData 2017, Boston,A, USA, December 11-14, 2017*. IEEE Computer Society, 2462–2471.

[15] Thomas A. Limoncelli. 2019. SQL is no excuse to avoid DevOps. *Commun. ACM* 62, 1 (2019), 46–49. https://doi.org/10.1145/3287299

[16] Dien-Yen Lin and Iulian Neamtiu. 2009. Collateral Evolution of Applications and Databases. In *Joint Intl. Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol)*. 31–40.

[17] Petros Manousis, Panos Vassiliadis, Apostolos V. Zarras, and George Papastefanatos. 2015. Schema Evolution for Databases and Data Warehouses. In *5th European Summer School on Business Intelligence , eBISS 2015 (Lecture Notes in Business Information Processing)*, Vol. 253. Springer, 1–31.

[18] Petros Manousis, Apostolos V. Zarras, Panos Vassiliadis, and George Papastefanatos. 2017. Extraction of Embedded Queries via Static Analysis of Host Code. In *29th International Conference on Advanced Information Systems Engineering (CAiSE 2017), Essen, Germany, June 12-16, 2017, Proceedings*. 511–526.

[19] Robert Martin. 2002. *Agile Software Development, Principles, Patterns, and Practices*. Pearson.

[20] Andy Maule, Wolfgang Emmerich, and David S. Rosenblum. 2008. Impact analysis of database schema changes. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn (Eds.). ACM, 451–460.

[21] Tom Mens, Maëlick Claes, Philippe Grosjean, and Alexander Serebrenik. 2014. Studying Evolving Software Ecosystems based on Ecological Models. In *Evolving Software Systems*, Tom Mens, Alexander Serebrenik, and Anthony Cleve (Eds.). Springer, 297–326.

[22] MEJ Newman. 2005. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics* 46, 5 (2005), 323–351. https://doi.org/10.1080/00107510500052444

[23] George Papastefanatos, Panos Vassiliadis, Alkis Simitsis, and Yannis Vassiliou. 2010. HECATAEUS: Regulating schema evolution. In *ICDE*. 1181–1184.

[24] Dong Qiu, Bixin Li, and Zhendong Su. 2013. An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications. In *2013 9th Joint Meeting on Foundations of Software Engineering ((ESEC/FSE))*. 125–135.

[25] Stefanie Scherzinger, Wolfgang Mauerer, and Haridimos Kondylakis. 2021. DeBinelle: Semantic Patches for Coupled Database-Application Evolution. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2697–2700.

[26] Stefanie Scherzinger and Sebastian Sidortschuck. 2020. An Empirical Study on the Design and Evolution of NoSQL Database Schemas. In *Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings (Lecture Notes in Computer Science)*, Gillian Dobbie, Ulrich Frank, Gerti Kappel, Stephen W. Liddle, and Heinrich C. Mayr (Eds.), Vol. 12400. Springer, 441–455.

[27] Robert E. Schuler and Carl Kesselman. 2019. A High-level User-oriented Framework for Database Evolution. In *31st International Conference on Scientific and Statistical Database Management, SSDBM 2019, Santa Cruz, CA, USA, July 23-25, 2019*. ACM, 157–168.

[28] D. Sjøberg. 1993. Quantifying schema evolution. *Information and Software Technology* 35, 1 (1993), 35–44.

[29] Ioannis Skoulis, Panos Vassiliadis, and Apostolos V. Zarras. 2015. Growing up with stability: How open-source relational databases evolve. *Information Systems* 53 (2015), 363–385.

[30] Michael Stonebraker, Raul Castro Fernandez, Dong Deng, and Michael L. Brodie. 2017. Database Decay and What To Do About It. *Commun. ACM* 60, 1 (2017), 11. https://doi.org/10.1145/3014349

[31] Uta Störl, Meike Klettke, and Stefanie Scherzinger. 2020. NoSQL Schema Evolution and Data Migration: State-of-the-Art and Opportunities. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang (Eds.). OpenProceedings.org, 655–658.

[32] Dennis Tsichritzis and Anthony C. Klug. 1978. The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Dabatase Management Systems. *Inf. Syst.* 3, 3 (1978), 173–191. https://doi.org/10.1016/0306-4379(78)90001-7

[33] Panos Vassiliadis. 2021. Profiles of Schema Evolution in Free Open Source Software Projects. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 1–12.

[34] Panos Vassiliadis, Michail-Romanos Kolozoff, Maria Zerva, and Apostolos V. Zarras. 2019. Schema evolution and foreign keys: a study on usage, heartbeat of change and relationship of foreign keys to table activity. *Computing* 101, 10 (2019), 1431–1456.

[35] Panos Vassiliadis and Apostolos V. Zarras. 2017. Schema Evolution Survival Guide for Tables: Avoid Rigid Childhood and You're En Route to a Quiet Life. *Journal of Data Semantics* 6, 4 (2017), 221–241.

[36] Panos Vassiliadis, Apostolos V. Zarras, and Ioannis Skoulis. 2017. Gravitating to rigidity: Patterns of schema evolution - and its absence - in the lives of tables. *Information Systems* 63 (2017), 24–46.

[37] Shengfeng Wu and Iulian Neamtiu. 2011. Schema Evolution Analysis for Embedded Databases. In *2011 IEEE 27th International Conference on Data Engineering Workshops (ICDEW '11)*. 151–156.