Parallel lives diagrams for co-evolving communities and their application to schema evolution*

Fanis Giachos[1], Nikos Pantelidis[2], Christos Batsilas[3], Apostolos V. Zarras[4] and Panos Vassiliadis[4]

[1]Piraeus Bank, Athens, Greece
[2]CGI Nederland, Rotterdam, Netherlands
[3]Natech S.A., Ioannina, Greece
[4]University of Ioannina, Ioannina, Greece

Abstract

In this paper, we address the problem of modeling co-evolving peers in communities over time. Our motivation comes from the area of software and schema evolution; however, we generalize our modeling to cover communities of peer entities in general, evolving over discrete time beats, with quantifiable measurements of behavior. Furthermore, we demonstrate how our modeling can facilitate the visualization, comprehension, and automated analysis of the lives of such communities.

Keywords

Evolving communities, Software Evolution, Schema Evolution, Software Visualization

1. Introduction

Software systems are never complete or perfect; hence they continuously evolve, in order to accommodate new requirements, adapt to changing operational environments, as well as to correct internal problems and errors, either prior, or after they are discovered at their usage. The study of software evolution has two aspects, as [1] eloquently states: (a) the what and why of software evolution, that "focuses on the properties of the phenomenon, its causes and identification of the drivers underlying development and maintenance activity", and, (b) the how, that is "the methods, tools and technology to facilitate disciplined and efficient software change". Understanding laws and patterns that guide software evolution allows us to recognize mechanisms, tendencies, and (ideally) deterministic behaviors of how software systems change.

A specific aspect we are addressing in this paper, concerns the understanding of how different parts of a software system evolve together. The parts of a software system behave as peers that co-exist in a community, where all the components must collaborate towards providing the necessary functionality. In particular, we are motivated by the study of schema evolution,

ER2023: Companion Proceedings of the 42nd International Conference on Conceptual Modeling: ER Forum, 7th SCME, Project Exhibitions, Posters and Demos, and Doctoral Consortium, November 06-09, 2023, Lisbon, Portugal

*The work of all authors has been conducted during their time in the Univ. of Ioannina

fgiahos@hotmail.gr (F. Giachos); pantelidis.nikos@outlook.com (N. Pantelidis); christosbats@gmail.com (C. Batsilas); firstname.lastname@cs.uoi.gr (A. V. Zarras); firstname.lastname@cs.uoi.gr (P. Vassiliadis)
0000-0001-9521-5853 (A. V. Zarras); 0000-0003-0085-6776 (P. Vassiliadis)

CEUR Workshop Proceedings (CEUR-WS.org)

where the components of a relational schema evolve over time to accommodate changing information needs by the surrounding applications. Overall, the research question that drives us is: *can we model, and trace, the information necessary to allow us to study the joint co-evolution of different parts of a software system (and, in particular, a relational schema) in order to be able (a) to understand how the different parts of a system co-evolve over time, (b) identify highlights and patterns over this co-evolution, and, (c) be able to come up with automated discoveries and reporting of significant findings over the studied histories?*

Example. To explicate our stance on the problem, we motivate the discussion with an example. In Figure 1 we depict a visual representation of the parallel lives of the relations of a relational database schema, which form a community of co-evolving entities.

Time is represented as a timeline of discrete time-beats (the columns of the visual representation). The different entities of the community (in the case of schema evolution: the relations of the schema) are visually represented as the rows of a two-dimensional matrix. Each entity has (a) a dedicated row where its life is visually depicted, and, (b) several aggregate details (depicted as a pop-up window) at the bottom of the figure. The most fundamental properties for an entity are (a) the timepoint when it joins the community (to which we typically refer to as "birth"), (b) a possible timepoint when it leaves the community (referred to as "death" – in our case, the table is removed from the schema), and, (c) the amount of change that takes place at each time point, along one or more quantifiable measurements (visually depicted via the color saturation in the diagram of Figure 1). The entities in this particular representation have been sorted according to their birth, to facilitate a visual understanding of the progression of events (other types of sorting, with different visual goals, are also possible – e.g., along the lines of the amount of change they have undergone). The core of the model is a two-dimensional matrix of $Entities \times TimeBeats$ that concentrates the information needed for representing, visualizing, studying and analyzing the history of the community.

In summary, in Figure 1, the usage of the aforementioned modeling along a timeline, a set of peer entities, and quantifiable measures of activity for each of them, is vividly demonstrated. The most obvious usage of the model is the visualization part, which allows a quick understanding (and reporting) of how the life of the community has evolved. Apart from facilitating reporting and understanding, the modeling allows the fully automated identification of highlights, or patterns, over the two-dimensional matrix: massive births, massive updates, progressive expansion, entities with continuous change, or entities without any change whatsoever, can be automatically discovered, reported and visually highlighted on the basis of our model.

Generalization. We have worked with the study of schema evolution histories at very large numbers. However, we claim that our results are generalizable to larger settings, beyond schema evolution. For a community to abide by our model one needs to have (a) a notion of time, in a timeline of discrete time steps; (b) a set of discrete entities that form a community; (c) the notion of entities joining and leaving the community (birth and death in our terminology) during the monitored timeline; (d) measurable quantities for the entities of the community that are measured throughout their participation in the community. Whenever the aforementioned properties hold, our framework covers the evolution of such a community and can provide the necessary modeling, visualization and analysis means to the analyst who wishes to understand how the community evolves. The components of a software project (be it packages, classes, modules, libraries or other software entities) are a clear case where our framework is directly

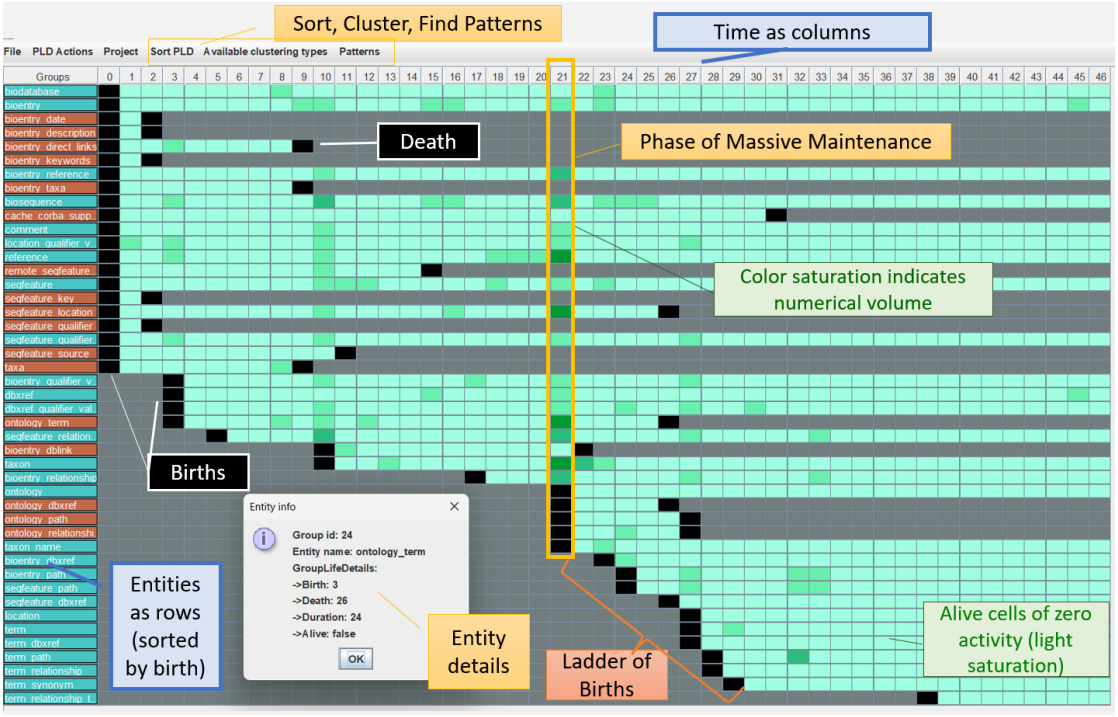

Figure 1: An annotated Parallel Lives Diagram, depicting the timeline, the entities, the changes, important events and details for the lives of the entities of a community – in this case, the relations of the schema of a database

applicable.

Contributions and Roadmap. After starting with the presentation of related work in Section 2, the paper proceeds to provide the following contributions. In Section 3, we present the conceptual model of our approach. We require the collaboration of Entities, Timelines, Measurement Types and Parallel Lives Matrices for the representation of the necessary information required to characterize how the peers of a community co-evolve. In Section 4, we present the definitions, examples and algorithms for the mining of interesting patterns from the co-evolution of the entities of a community. In Section 5, we present how the application of our modeling to the case of schema evolution, revealed interesting patterns of change over a data set of 195 schema histories of Free-Open Source projects. Finally, we conclude our deliberations with a discussion of the contributions of this paper as well as open roads for future work.

2. Background

To the best of our efforts, we were unable to find any works on modeling and visualizing co-evolving communities, in general. However, there are several works pertaining to software and database evolution that are clearly the motivation for our work – although we argue that our modeling can be generalized to a broader set of contexts.

Software Evolution. Software evolution has been studied for decades at several levels:

software architecture [2], design [3] and implementation [4]. The main driver for studying software evolution have been Lehman's laws and the theory that accompanies them, starting in the mid '70's all the way to nowadays. For a discussion of Lehman's laws on can refer to [5, 6, 7, 8, 9], summarized in [1]. Other attempts towards finding regularities and patterns in software evolution include [10, 11, 12, 13, 14, 15, 16, 17, 18, 17, 19, 20]. Although not all laws are considered valid any more, the idea for searching in patterns on how software evolves is fundamental in the research on software evolution.

Schema Evolution. Schema evolution, which has been the main driver of our research, involves the progressive change of the internal structure of a database over time. Typically, the studies in the area of schema evolution are mostly observational, assessing the qualitative and quantitative characteristics of schema evolution – i.e., answering the question "what are the characteristics of the phenomenon that we study?". Several studies address this question in the field of relational databases [21], [22],[23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35]. Currently, interest has passed from relational to non-relational databases [36], [37] – see [38] for an overview.

Schema and Source co-evolution. As the impact of schema evolution can be very large to the ecosystem built around the database [39, 40], there are also studies concerning ways to adapt queries whenever the schema changes [41], [42], [43], [44], [45] – see [46] for an overview. In the meanwhile, there are also works on how schema and source code co-evolve, both on the area of studying the joint evolution and in the area of proposing techniques to synchronize schema and applications as they both change [21], [23], [24], [25], [47], [48], [49].

Software visualization of co-evolution. To a large extent, the most related area of research to our effort is software visualization, with an emphasis on tracing co-evolution. The papers in this category are not explicitly providing a conceptual modeling perspective to their method, but rather, they focus on the visualization aspects.

The authors of [50] provide some first visualization techniques for detecting patterns of change. The authors of [51] provide parallel timelines resembling violin charts, for different parts of the code to describe important events in the history of a system. In [52], the authors provide a system with dense evolution lines for different parts of the code, where notes can be attached for commenting. The authors of [53] use heatmaps to demonstrate how classes cooperate in use cases, or how much each developer contributes to the maintenance of a system. The authors of [54] propose an Entities $\times$ Entities matrix to trace co-evolution of code components (in contrast to our proposal that includes time, too). [55] visualizes the co-change of code and tests via bubble-charts and [56] as addition-deletion bar-charts.

For 3D visual representations of evolving software, quite often represented via the "city metaphor", one can refer to [57], [58], [59], [60]. We avoid the 3D city metaphor as overly complicated contrasted to the simple representation that we provide.

3. The model

The main concepts that define the landscape of communities of jointly evolving peers concern (a) time, (b) peers, and, (c) measured behavior.

3.1. Basic concepts

We assume a linear discrete version of time. We consider as time, a domain of values that is isomorphic to the non-negative integers and consists of discrete and equidistant time beats. This is not necessarily restricted to "human" time (which, of course, is also eligible for the role). For example, when we study evolving software repositories, time can be modeled as the commits made by the developers to a branch of the repository; in this case, each commit is treated as a time beat and commits are isomorphically mapped to non-negative integers.

We also assume a community of peer entities that evolve together. The entities can be arbitrary. In the case of schema evolution, the entities under investigation are the relations that appeared in the entire history of a certain schema. For each entity we have (a) a time-beat of birth where the entity joins the community (for the case of schema evolution: the table is introduced in the schema), (b) a time-beat of death, when the entity leaves the community (for the case of schema evolution: a table is deleted from the schema), and, in-between, (c) an evolving behavior, characterized through a vector of measurements (for the case of schema evolution: a vector of measurements quantifying, for each table, for each time-beat, the number of attributes inserted into the table, ejected from it, modified with respect to their data type, etc).

Therefore, the main concepts appearing in the domain can be listed as follows:

- **Timeline**: a linear domain $\mathcal{T}^{\infty}$, which is isomorphic to the non-negative integers $\mathbb{N}^0$ and provides a common context (or, timeframe) for the evolution of a community of peer entities. For practical purposes we will work with finite histories, thus, time will be a finite subset of $\mathcal{T}^{\infty}$, $\mathcal{T} = \{T_0, \ldots, T_m\}$.
- **Beat**: A beat is a unique member of a time domain. Thus, it can be a time unit (second, day, month, etc) or anything simulating a time domain (a stock market working day, a commit in a software repository, etc).
- **Entity**: A distinct member of a community whose life is being monitored. An entity E has a time point of *appearance*, $E.T^a$, when it first joins the community, an optional time point of ultimate *disappearance* $E.T^d$, when it leaves the community and at any time point t_i it has a state (which will be presented in the sequel).
- **Community**: a finite set of Entities, $\mathcal{C} = \{E_1, \ldots, E_n\}$ monitored together for their joint evolution. A set of stocks in a stock market, or the set of tables of a schema are examples of communities.
- **Measurement Type**: A common quantity that is monitored for the lives of the entities of a community and evaluated through numeric measurements. The entities of a community can be monitored for a number of Measurement Types. For example, a table can be monitored for the number of attributes injected, ejected, having their data type updated, as well as the sum of the above as a measurement of total activity.Each of these quantities is a Measurement Type. We assume that the entities of a community are all monitored upon a common set of Measurement Types, $\mathcal{M} = \{M_1, \ldots, M_k\}$, with each measurement type M_i having as its domain of values $dom(M_i)$, which, for simplicity, we will uniformly assume to be $\mathbb{R}$. Every member of the domain of a Measurement Type is a **Measurement**.
- **TimeEntityMeasurementSet(TEM):** The combination of an entity, a beat, and a vector of measurements - i.e., a unique point in the life of an entity, along with the measurements

that pertain to it. Assuming we have fixed the set of measurement types into a single Measurement Type, we refer to a **TimeEntityMeasurement** object. Thus, $TEM^{\mathcal{M}}$ is a function $TEM^{\mathcal{M}}$: $\mathcal{T} \times \mathcal{C} \rightarrow \mathcal{M}$ and TEM^{M} for a single measure M is a function TEM^{M}: $\mathcal{T} \times \mathcal{C} \rightarrow M$ (practically projecting $\mathcal{M}$ to M).

How are all these concepts combined? We introduce the **Parallel Lives Matrix (PLM)** which is a matrix having (a) all the entities of a community as rows, (b) all the beats of a time domain for this community as columns, (c) the respective TEM objects as cells. Although we will revisit the definition in the sequel, for the moment we can point to Figure 2 that depicts a visual representation of a PLM with a single Measure Type, $TotalActivity$.

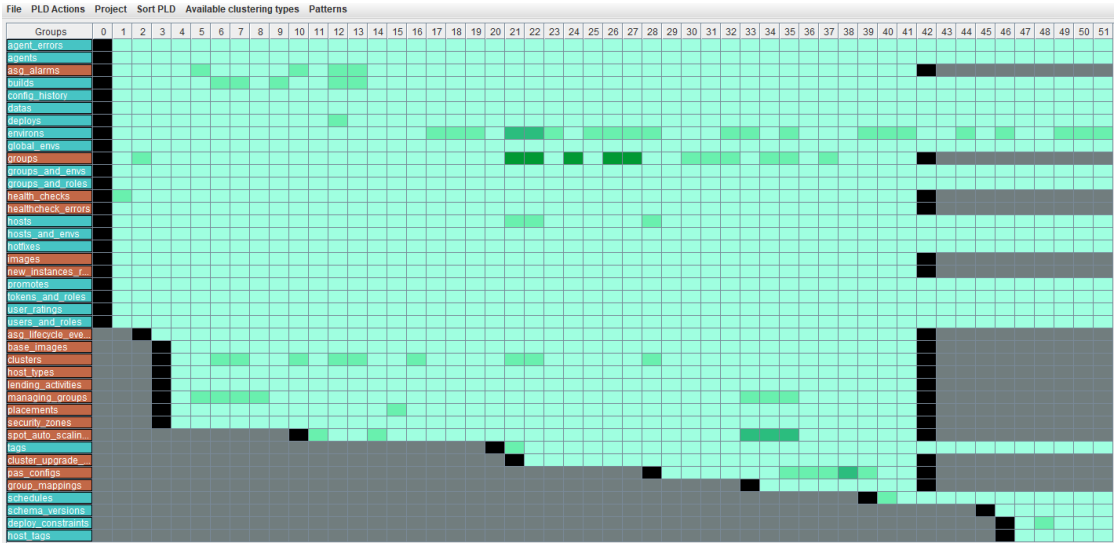

Figure 2: A visual depiction of the Time Entity Measurements for a community of peers (the tables of a database schema) for a time line of commits in a public repository, over a single measure: total change activity.

3.2. Groups of time beats and entities

As already mentioned, one of the main purposes for working with this representation of the lives of peers in a community is the possibility of visualizing the entire history of the community in a two-dimensional surface, like a screen. The two-dimensional visual representation of a Parallel Lives Matrix as a table of columns and rows is a straightforward, simple and intuitive solution. However, throughout the construction and subsequent usage of a system to perform this visualization, we repeatedly came across a problem, which seems to be fundamental in handling long histories and large communities: Time and again, either the time-line, or the community size was too big for a screen to accommodate. A generic requirement, thus, occurred, to be able to group entities into homogeneous groups and time beats into homogeneous phases, in order to reduce the visual footprint of the matrix. The following concepts, are therefore added to our conceptual model for the handling of the evolving lives of peer communities.

- **Phase**: A phase P is a list of consecutive time beats P=$\{T_{start} \dots T_{end}\}$ in the same domain. A time beat is also a trivial case of a phase. Practically, phases allow us to zoom out time in coarser time granules and e.g., group beats in months instead of individual days, in order to make the visualization fit in the limited area of a screen. A **Phased Timeline** $\mathcal{P}^{\mathcal{T}}$, or simply $\mathcal{P}$, over a simple Timeline $\mathcal{T}$ is a list of phases $\mathcal{P} = \{P_1, \dots, P_\tau\}$ that introduces a partition over $\mathcal{T}$, i.e., $\mathcal{T}$ is fully covered by $\mathcal{P}$, and all members of $\mathcal{P}$ are pairwise disjoint.
- **Entity Group**: Assuming a community $\mathcal{C}$, an entity group $G = \{E_{x1}, \dots, E_{xk}\}$ is a subset of $\mathcal{C}$. Entity groups are produced by clustering entities with similar lives, to reduce the amount of rows in our visualization. A single entity is a trivial case of an entity group. A **Grouped Community** $\mathcal{G}^{\mathcal{C}}$, or simply $\mathcal{G}$, is a partition of a community $\mathcal{C}$ into pairwise disjoint and fully covering grouped communities.
- **GroupPhaseMeasurementSet (GPM):** A **GroupPhaseMeasurementSet** is defined with respect to the combination of an entity group and a phase; its role is to aggregates the TimeEntityMeasurementSet instances pertaining to the entities of the entity group, and the time beats of the phase.

Thus, assuming an aggregate function γ, and a single measure type M, GPM^M is a function GPM^M: $\mathcal{P} \times \mathcal{G} \rightarrow M^\gamma$, where M is mapped to a new Measure Type M^γ, s.t., if for a given phase P, and a given entity group G, $GPM^M(P, G) = v$, then v is the aggregation of all m_i, s.t., $m_i \in TEM^M(T_j, E_k), T_j \in P$ and $E_k \in G$. Then, $GPM^{\mathcal{M}}$ is a function $GPM^{\mathcal{M}}$: $\mathcal{P} \times \mathcal{G} \rightarrow \mathcal{M}^\gamma$, where $\mathcal{M}^\gamma$ is produced by the Cartesian Product of all M_i^γ, for all $M_i \in \mathcal{M}$.

Different techniques are applied to perform the groupings of time beats and entities. Specifically, a time-clustering algorithm splits the time domain into disjoint, consecutive phases that fully cover the original time domain, with the goal of retaining as much uniformity in terms of activity within each phase. Exactly along the same line, an entity-clustering algorithm splits a set of peers into a set of disjoint clusters that fully cover the original set of entities, with the goal of retaining as much uniformity in terms of activity within each group. For example, in our implementation, we create phases and entity groups using agglomerative clustering over the total activity of beats and entities respectively (with the observation that when it comes to time, the beats of a cluster must be consecutive).

Remember also that single beats and single entities are trivial phases and entity groups; therefore, in the absence of clustering, TEM's are trivial GPM's and can be treated as such.

Now, we can revisit the definition of a PLM, to generalize it to Phases and Entity Groups. A **Parallel Lives Matrix (PLM)** which is a matrix having (a) all the entity groups of a community as rows, (b) all the phases of a time domain for this community as columns, (c) the respective GPM objects as cells. We will employ the notation PLM^0 whenever both entity groups and phases are trivial, and therefore the PLM concerns individual entities and time beats.

A **Parallel Lives Diagram (PLD)** is a diagram that visually represents a PLM for a single measurement type. Practically, a Parallel Lives Diagram is the visual representation for a GPM^M.

In Figure 3, we depict the basic modeling notions of our approach.

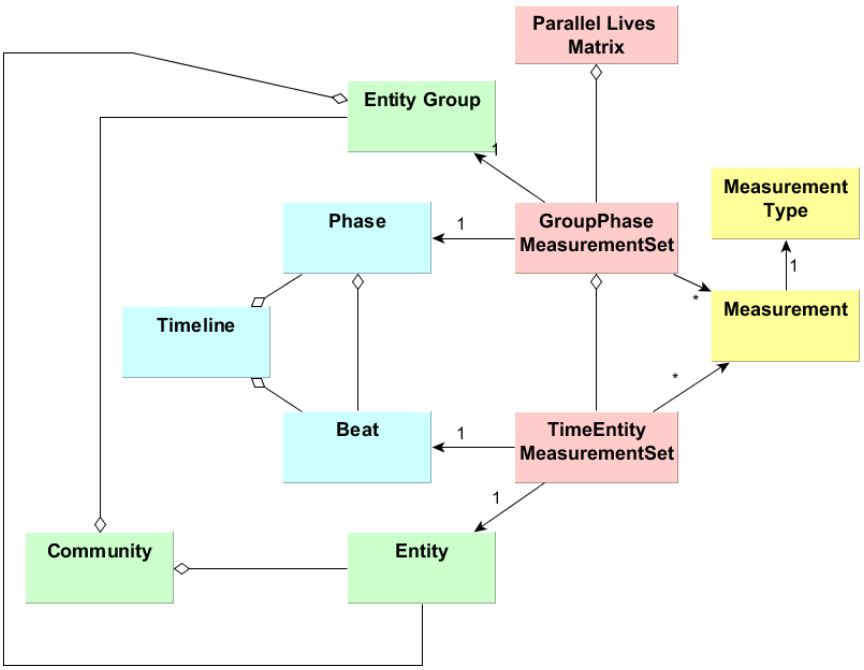

Figure 3: The model for the Parallel Lives Matrix environment

3.3. Cell states

By monitoring the community on-line, or, by studying log files post-hoc, we can have a way to know, for each time beat, for each entity, whether this entity was a member of, or had left, the community, and, in the case where it was a member of the community, the specific measurements that pertain to the monitored measurement types (which we will collectively refer to as "activity").

Given this knowledge, concerning presence and activity, every cell in a PLM^0 (equivalent: PLD), $PLM^0[E, T]$ can have one of the following *states* (remember that an entity can leave and rejoin many times):

- *Active*: the entity group E is *alive*, i.e., it has appeared in a previous beat than T and is still a member of the community.
- *Absent* or *Inactive*: the entity E has not been created yet at time T, or has been deleted in a previous beat than T and not recreated at, or, before T (remember there are entities that leave the community and later re-join).
- *Birth* or *Appearance*: the cell is *Active* and this is the first cell of the row corresponding to E with status *Active* (i.e., this is the first appearance ever of the entity in the community).
- *Rebirth*: the cell is not in a *Birth* state, however, it is *Active* and its previous cell in the same row is *Absent*.
- *Disappearance*: the cell is *Active* and the next cell of the row is *Absent*.

- *Death*: the cell is in a state of *Disappearance* and there is no other birth of the entity later – equivalently, it the last beat where the entity is active, and it is followed by a contiguous period of beats, spanning all the way to the final beat of time, where the cell state is absent.

Aggregation. Whereas the state of non-aggregate data is straightforward to obtain, the same does not hold for the state of aggregate cells. Assume we merge two entities E_1 and E_2 into a group G, while at the same time, a list of their beats are merged into a new phase P. Then, we have a new cell $PLM[G, P]$ whose state we need to determine. Recursively, the problem generalizes into merging a window of the PLM into a single cell.

There are several possibilities for this decision. One possibility is to prioritize states: for example, one might say that births are more important than alive states, which are more important than disappearance states. This is an arbitrarily set order, for exemplary reasons – one can allow other state rankings, depending on individual preferences. Another possibility involves taking the state of the first / median / last cell of the first / middle / last entity of the merged window. A majority vote of the cells is a third possibility.

4. The patterns

The power of our modeling is based on its simplicity. The model constructs are amenable to a straightforward visualization, via a direct mapping to a two-dimensional matrix. Apart from the obvious benefits in terms of intuition that come with simple visualizations, we have taken the opportunity to mine for *patterns* over the representation. Patterns are interesting properties of the two-dimensional representation, possibly on the basis of a single column, row, or cell – and potentially via combinations of them – that demonstrate interesting behavior for the purpose of understanding recurring behaviors in the lives of communities.

4.1. Example

In Figure 4, we can observe the existence of several patterns, concerning massive births deaths, updates, and, progressive expansion. For the non-colorblind readers, the color of the cells per pattern is also reported.

- Observe the existence of a "Multiple Birth Stairs" pattern (a) between columns 0 and 5 and (b) between columns 20 and 29. In both cases, there are consecutive columns with cells of state Birth, and the number of these cells is higher than the threshold. The involved cells are painted pink.
- In column 2, there are bulk deletions of entities (i.e., there are more than the threshold, with a number greater than 3), so the column supports the "Multiple Deletion" pattern. The involved cells are painted red.
- Similarly, the "Multiple Updates" pattern is supported by columns 3, 10, 21, 23, 27 and 32. The involved cells are painted yellow.
- Finally, in column 0 it is easy to observe the "Multiple Births" pattern. Color-wise, the cell's coloring is overridden by the color of the "Multiple Birth Stairs" pattern.

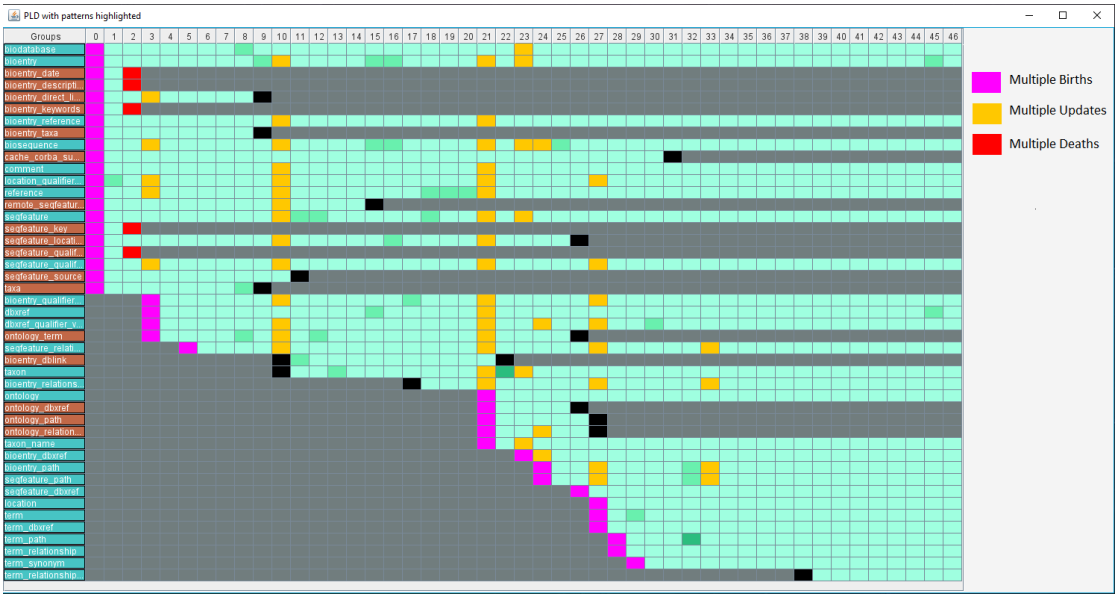

Figure 4: The PLD for Biosql with the cells participating in the patterns colored.

4.2. Pattern definitions

In this section, we introduce a set of pattern families, as well as concrete patterns that belong to them that are possibly derivable from the information on the evolution of a community. Both the set of families and the set of patterns are extensible; in our deliberations we refer only to the ones that we have implemented.

As already mentioned, the tool we use for registering the state and evolution of the population of peer entities is a two-dimensional matrix $Beats \times Entities$, which can easily be grouped into a matrix $Phases \times EntityGroups$. In the rest of our deliberations, we will use the latter as our setup of reference; however, all the characterizations and algorithms are immediately applicable to the simple domain model, which is a trivial case of the latter setup.

The first family of patterns that we introduce (Def. 4.1) involves patterns where each column can be tested in isolation from the others. This can involve the existence of a cell with a certain state ("did a birth occur in this phase?"), or, more commonly, whether the cardinality of cells with a certain state exceeds a threshold value ("there are too many births in this phase"). The latter sub-family of patterns is singled out also as a distinct family of interest (Def. 4.2).

Definition 4.1. ***Single column, local-cell-test pattern.*** *Assume a $PLD[n \times m]$, with n rows and m columns. A single column, local-cell-test pattern is a predicate that when applied to a column, returns true or false on the basis of evaluating a condition on the cells of the column, one-at-a-time, i.e., independently of the state of other cells or columns.*

Definition 4.2. ***Single column, counting, local-cell-test pattern.*** *A single column, counting, local-cell-test pattern is a single column, local-cell-test pattern, where the verification of pattern existence involves counting the number of cells of a certain state.*

For a column C, and the possibility of testing the state of any of its cells, say c, independently of other cells or columns, we can have a counting pattern test of the form:

$$countingPatternTest(C, \tau) = \{count(c) > \tau | c \in C, holdsTestedState(c)\}$$

where τ is a counting threshold. Three prominent examples of such a pattern concern:

- $multipleBirths(C, \tau)$, the case of multiple births in a column, with $holdsTestedState(c)$: $c.state == birth$
- $multipleDeaths(C, \tau)$, the case of multiple deaths in a column, with $holdsTestedState(c)$: $c.state == death$
- $multipleUpdates(C, \tau)$,the case of multiple updates in a column, with $holdsTestedState(c)$: $c.state == active \;\&\&\; c.updateMetric > 0$

Another family of patterns involves sliding a window over (a) several columns, and, (b) several rows of the PLM ("several" can become "all" to capture holistic patterns, too) and testing a predicate (def 4.3). A pattern that we have frequently observed in the evolution of relational schemata involves subsequent births in contiguous columns (Def. 4.4). Observe Figure 4: the PLD of the figure shows the life of the schema of a specific database-backed project, named BioSQL. The rows of the PLD are sorted by birth (columns are inherently sorted as they represent time and they are isomorphic to the natural numbers). Observe the middle bottom part of the figure: several adjacent columns demonstrate births, one after the other. To the extent that the PLD is sorted by birth and time, the visual impression from the respective birth cells (highlighted in intense tonality – for the non-colorblind: in pink) is a "staircase" which is also the name of the pattern.

Definition 4.3. ***Sliding Window pattern***. *Assume a $PLD[n \times m]$, with n rows and m columns. A sliding window pattern is a predicate that when applied to a set of columns, returns true or false on the basis of evaluating a condition over the entire set of cells contained in a "window" area defined over several (possibly all of the) rows of the involved columns.*

Definition 4.4. ***Strict τ^B-staircase of births***. *Assume a τ^B-sized list $L_j^{\tau^B}$ of adjacent columns $L_j^{\tau^B} = \{C_j, C_{j+1}, \ldots, C_{j+\tau^B}\}$. If every column in the list contains cells with a birth status, the list $L_j^{\tau^B}$ demonstrates a strict staircase of births.*

We can relax this definition by allowing some of the list's columns not to contain births (you can see in the middle bottom part of Figure 4 a couple of such columns that do not annul the overall behavior -or the visual impression- of a staircase). One potential definition (admittedly, approximate) of a relaxed staircase pattern is based on simple counting cells with birth status.

Definition 4.5. ***Approximate $\tau^{B,N}$-staircase of births***. *Assume a τ^B-sized list of adjacent columns $L_j^{\tau^B} = \{C_j, C_{j+1}, \ldots, C_{j+\tau^B}\}$. If the columns in the list contains at least τ^N cells with a birth status, the list $L_j^{\tau^B}$ demonstrates an approximate staircase of births.*

Algorithm 1: Generic single-column, count-based pattern extractor algorithm.

Input: a matrix $PLD[n \times m]$, with n entity groups, m phases, and $PLD[E_i, P_j]$ the amount of change that took place for entity group E_i at phase P_j; a threshold of occurrences that qualifies a column to fulfill a pattern τ

Output: a set of columns $\mathbf{C}$, each of which demonstrates an occurrence of the tested pattern

```
1  begin
2      C = ∅
3      forall P_j ∈ 𝒫 do
4          counter = 0
5          forall G_i ∈ 𝒢 do
6              if supportsPattern(PLD[G_i, P_j]) then
7                  counter++
8              end
9          end
10         if counter > τ then
11             C = C ⋃ P_j
12         end
13     end
14     return C
15 end
16 Interface supportsPattern(cell) : Boolean is an overloaded interface
17     test cell state depending on the pattern searched
18 end
```

4.3. Algorithm for testing single-column, counting-based patterns

The basic algorithm for the handling of single-column patterns where the qualification of a column for supporting the pattern is based only on counting, is depicted in Algorithm 1. For different single-column patterns, different conditions and thresholds can apply. In our deliberations, we have worked with τ having a value of 3. Depending on the pattern being searched, the implementations of the function $supportsPattern(cell)$ differ. Specifically:

- For the detection of massive births in a column, the test requires *Cell.state == BIRTH*
- For the detection of massive deaths in a column, the test requires *Cell.state == DEATH*
- For the detection of massive updates in a column, the test requires *Cell.state == ACTIVE && $PLD[G_i, P_j] > 0$*

4.4. Variations and optimizations

There are several variations that one can apply to the simple pattern checking algorithm. A first, simple modification can be applied when testing rows of the PLD instead of columns. In

this case, instead of searching for phases where a pattern emerged, one can search for entity groups with interesting behavior. Examples of such tests include:

- Tests for rows, with a massive/low/zero number of changes (i.e., the ones whose activity is beyond/below a certain threshold)
- Tests for rows with more than one birth, i.e., entities that joined the community, left, and re-joined later (e.g., in the case of schema evolution, tables that were removed from the schema, only to reappear a few commits later)

A straightforward optimization that we have performed in our implementation is to embed all single-column checks into the same nested-loops pair. Thus, instead of checking the *supportPattern* predicate for different patterns in separate loops, we execute all the *supportPattern* checks, for all the patterns we want to test, within a single loop, via different counters and result column-sets, one per pattern. Note also that for the patterns that we check, the matrix need not be sorted, as the checks are based only on counting cells with the appropriate state.

4.5. Multi-column, shape-based patterns

Now we can define an algorithm for the relaxed version of the birth staircase. For every column of a sorted PLD, Algorithm 2 checks a window of τ^W columns for births. To the extent that the PLD is sorted by birth, if these columns contain births, these births will be placed immediately after the last birth of the column under investigation. The algorithm is approximate, as, instead of a shape-based pattern, it checks for the cardinality of the set of cells with births in the window. If this set exceeds a threshold τ^N, the window qualifies for a staircase.

5. An application to the study of schema evolution

In this Section, we discuss how our framework is applied to the study of schema evolution. In particular, we investigate the existence of the aforementioned patterns in the histories of relational schemata from Free-Open Source projects in a large dataset from the literature.

5.1. Dataset and toolset

For investigating the extent of the presence of patterns, we employ the Schema_Evo_2019 data set[1] from the literature; specifically, from [34]. The data set contains 195 schema histories of Free-Open Source projects, that were collected from Github with specific collection criteria. In order to avoid bias, as well as insignificant projects, the author of [34] filtered out of the corpus the projects with 0 stars or just 1 contributor, DDL files with 'example', 'demo', 'test', terms in their path, and, projects without a history of versions for the DDL file. We refer the reader to [34, 35] for a detailed discussion of the collection process, its representativeness, and its generalization liabilities. The time is measured in commits, the entities are the individual tables that appear in the schema histories and the measures monitored are: attributes born

[1]Available at Github at https://github.com/DAINTINESS-Group/Schema_Evolution_Datasets/tree/master/SchemaEvolutionDatasets2020

Algorithm 2: Relaxed birth staircase algorithm

Input: a matrix $PLD[n \times m]$, with n entity groups, m phases, and $PLD[G_i, P_j]$ the amount of change that took place for entity group G_i at phase P_j; a threshold of occurrences that qualifies a column to fulfill a pattern τ^N; a column-width threshold of the window τ^W; a row-height threshold of the window τ^H;

Output: a set of columns $\mathbf{L}^T$, each of which demonstrates an occurrence of the tested pattern; a set of cells $\mathbf{L}^C$ participating in the pattern

begin
  Sort the rows of PLD by birth, ascending
  $\mathbf{L}^T = \emptyset$; $\mathbf{L}^C = \emptyset$
  forall $C_j \in PLD$ **do**
    Let $\mathbf{C}_j$ be the set of cells of column C_j with $state == BIRTH$
    Let $R_j^\star$ be the last row with a cell in a state of birth, at column C_j
    $\mathbf{L}_j^C = \emptyset$; $\mathbf{L}_j^T = \emptyset$ ▷ $\mathbf{L}_j^C$ `cell-set`, $\mathbf{L}_j^T$ `column-set, locally`
    ▷ `Iterate the window of col's post` C_j`, rows post` $R_j^\star$
    forall $C_k \in \{C_{j+1}, \ldots, C_{j+\tau^W}\}$ **do**
      forall $R_i \in \{R_{j+1}^\star, \ldots, R_{j+\tau^H}^\star\}$ **do**
        forall *cells* $c = PLD[R_i, C_k]$ **do**
          if $c.state == BIRTH$ **then**
            add c to $\mathbf{L}_j^C$
            add C_k to $\mathbf{L}_j^T$
          end
        end
      end
    end
    if $|\mathbf{L}_j^C \bigcup \mathbf{C}_j| > \tau^N$ **then**
      $\mathbf{L}^T = \mathbf{L}^T \bigcup \mathbf{L}_j^T$
      $\mathbf{L}^C = \mathbf{L}^C \bigcup \mathbf{L}_j^C \bigcup \mathbf{C}_j$
    end
  end
  return $\mathbf{L}^T$,$\mathbf{L}^C$
end

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 – all summarized in a measure of total activity (which is the one employed in the respective PLM's and PLD's).

We have implemented a tool[2] that allows the parsing, internal representation and analysis of community histories, which has been used to study evolving schema histories.

[2] https://github.com/DAINTINESS-Group/PlutarchParallelLives

Table 1
The descriptive statistics for the presence of patterns in the Schema_Evo_2019 data set.

	Total # Columns	Total # Rows	# Columns In Patterns	# Rows In Patterns	%Col's in patterns	%Rows in patterns	#Total Patterns	#Births Patterns	#Deaths Patterns	#Upd. Patterns	#Stairs Patterns
MAX	516	283	36	253	1	1	33	10	4	16	3
MIN	2	1	0	0	0	0	0	0	0	0	0
SUM	2796	2872	410	2545	38.30	112.04	319	148	26	97	48
AVG	14.34	14.73	2.10	13.05	0.20	0.57	1.64	0.76	0.13	0.50	0.25
COUNT	195	195	195	195	195	195	195	195	195	195	195
Median	**4.00**	**6.00**	**1.00**	**5.00**	**0.12**	**0.85**	**1.00**	**1.00**	**0.00**	**0.00**	**0.00**
StdDevP	44.61	30.33	3.95	29.01	0.22	0.46	3.12	1.03	0.50	1.62	0.57
Mode	2	1	0	0	0	0	0	0	0	0	0
Count Non Zero	**195**	**195**	**120**	**120**	**120**	**120**	**120 (62%)**	**109 (56%)**	**18 (09%)**	**41 (21%)**	**38 (19%)**

In Table 1, we depict the descriptive statistics for the Schema_Evo_2019 data set, with particular emphasis on the median and probability of presence for the discussed patterns. In all our experiments we have used a quite moderate threshold of $\tau = 3$. It is quite interesting that the patterns do not demonstrate a uniform behavior of presence. The Massive Birth pattern is fairly popular and present in 56% of the studied projects. This can be quite easily explained by the fact that most databases start with a 'big-bang' of introducing a significant percentage of their schema in the 0-th version. On the other hand, several patterns are rather unpopular: as typically mentioned in the literature, the removal of tables is scarce – let alone the massive removal (present in just 9% of the projects). The progressive expansion in subsequent steps is present in just 19% of the studied projects. Somewhere in the middle of the popularity spectrum is the existence of massive updates: in 21% of the projects, one can observe the presence of collective, focused maintenance, or expansion, of the schema.

6. Conclusions

In this paper, we have presented a conceptual model that involves entities, timelines, measurements, and their groupings in parallel lives matrices, in order to capture how the different entities of a community co-evolve. The model allows the visualization and understanding of the community evolution in a simple, but also powerful, way. The model also allows the mining of interesting patterns of change that highlight important points and members in the evolution of the community. We have applied our modeling to the case of schema evolution (which has been the motivating reason for this research) and derived patterns of change from a large number of schema histories.

There are several paths for future research. We have only scratched the surface of the patterns that can be investigated over Parallel Lives Diagrams. The generalization of birth staircases and massive updates to a "x changes soon after y" pattern is a simple example. Tool-wise, the interactive handling of roll-ups and drill-downs in the case of hierarchical structures is also a possibility. Finally, the fully automated reporting, that requires the ranking and pruning of the discovered patterns, in terms of their significance is another potential road for future research.

References

[1] M. M. Lehman, J. C. Fernandez-Ramil, Software Evolution and Feedback: Theory and Practice, Wiley, 2006. ISBN-13: 978-0-470-87180-5.

[2] M. Wermelinger, Y. Yu, A. Lozano, Design principles in architectural evolution: A case study, in: 24th IEEE International Conference on Software Maintenance (ICSM 2008), Beijing, China, 2008, pp. 396–405.

[3] Z. Xing, E. Stroulia, Analyzing the evolutionary history of the logical design of object-oriented software, IEEE Trans. Software Eng. 31 (2005) 850–868.

[4] I. Herraiz, D. Rodriguez, G. Robles, J. M. Gonzalez-Barahona, The evolution of the laws of software evolution: A discussion based on a systematic literature review, ACM Comput. Surv. 46 (2013) 1–28. doi:10.1145/2543581.2543595.

[5] L. A. Belady, M. M. Lehman, A model of large program development, IBM Systems Journal 15 (1976) 225–252.

[6] M. M. Lehman, Programs, life cycles, and laws of software evolution, Proceedings of the IEEE 68 (1980) 1060–1076. doi:10.1109/PROC.1980.11805.

[7] M. M. Lehman, Laws of software evolution revisited, in: Proceedings of 5th European Workshop on Software Process Technology, (EWSPT '96), Nancy, France, October 9-11, 1996, 1996, pp. 108–124.

[8] M. M. Lehman, J. F. Ramil, P. Wernick, D. E. Perry, W. M. Turski, Metrics and laws of software evolution - the nineties view, in: 4th IEEE International Software Metrics Symposium (METRICS 1997), 1997, p. 20.

[9] M. M. Lehman, J. F. Ramil, D. E. Perry, On evidence supporting the feast hypothesis and the laws of software evolution, in: 5th IEEE International Software Metrics Symposium (METRICS 1998), Bethesda, Maryland, USA, 1998, pp. 84–88.

[10] M. J. Lawrence, An examination of evolution dynamics, in: Proceedings, 6th International Conference on Software Engineering (ICSE 1982), Tokyo, Japan, 1982, pp. 188–196.

[11] S. S. Pirzada, A Statistical Examination of the Evolution of the Unix System, Ph.D. thesis, Imperial College, University of London, 1988.

[12] N. T. Siebel, S. Cook, M. Satpathy, D. Rodríguez, Latitudinal and longitudinal process diversity, Journal of Software Maintenance 15 (2003).

[13] M. W. Godfrey, Q. Tu, Evolution in open source software: A case study, in: Proceedings of the International Conference on Software Maintenance, 2000, pp. 131–142.

[14] M. W. Godfrey, Q. Tu, Growth, evolution, and structural change in open source software, in: Proceedings of the 4th International Workshop on Principles of Software Evolution, IWPSE '01, 2001, pp. 103–106.

[15] G. Robles, J. J. Amor, J. M. Gonzalez-Barahona, I. Herraiz, Evolution and growth in large libre software projects, in: Proceedings of the Eighth International Workshop on Principles of Software Evolution, IWPSE '05, 2005, pp. 165–174.

[16] S. Koch, Software evolution in open source projects: a large-scale investigation, J. Softw. Maint. Evol. 19 (2007) 361–382. doi:10.1002/smr.v19:6.

[17] G. Xie, J. Chen, I. Neamtiu, Towards a better understanding of software evolution: An empirical study on open source software, in: 25th IEEE International Conference on Software Maintenance (ICSM 2009), Edmonton, Alberta, Canada, 2009, pp. 51–60.

[18] I. Herraiz, G. Robles, J. M. Gonzalez-Barahon, Comparison between slocs and number of files as size metrics for software evolution analysis, in: Proceedings of the Conference on Software Maintenance and Reengineering, CSMR '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 206–213. URL: http://dl.acm.org/citation.cfm?id=1116163.1116405.

[19] R. Vasa, Growth and Change Dynamics in Open Source Software Systems, Ph.D. thesis, Swinburn Univ. of Technology, Australia, 2010.

[20] A. Israeli, D. G. Feitelson, The linux kernel as a case study in software evolution, J. Syst. Softw. 83 (2010) 485–501. doi:10.1016/j.jss.2009.09.042.

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

[22] C. Curino, H. J. Moon, L. Tanca, C. Zaniolo, Schema evolution in wikipedia: toward a web information system benchmark, in: Proceedings of ICEIS 2008, 2008.

[23] D.-Y. Lin, I. Neamtiu, Collateral evolution of applications and databases, in: Joint Intl. Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol), 2009, pp. 31–40.

[24] S. Wu, I. Neamtiu, Schema evolution analysis for embedded databases, in: 2011 IEEE 27th International Conference on Data Engineering Workshops, ICDEW '11, 2011, pp. 151–156.

[25] D. Qiu, B. Li, Z. Su, 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), 2013, pp. 125–135.

[26] A. Cleve, M. Gobert, L. Meurice, J. Maes, J. H. Weber, Understanding database schema evolution: A case study, Sci. Comput. Program. 97 (2015) 113–121.

[27] I. Skoulis, P. Vassiliadis, A. V. Zarras, Growing up with stability: How open-source relational databases evolve, Information Systems 53 (2015) 363–385.

[28] P. Vassiliadis, A. V. Zarras, I. Skoulis, Gravitating to rigidity: Patterns of schema evolution - and its absence - in the lives of tables, Information Systems 63 (2017) 24–46.

[29] P. Vassiliadis, A. V. Zarras, Schema evolution survival guide for tables: Avoid rigid childhood and you're en route to a quiet life, Journal of Data Semantics 6 (2017) 221–241.

[30] J. Delplanque, A. Etien, N. Anquetil, O. Auverlot, 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, 2018, pp. 635–644.

[31] P. Vassiliadis, M. Kolozoff, M. Zerva, A. V. Zarras, Schema evolution and foreign keys: a study on usage, heartbeat of change and relationship of foreign keys to table activity, Computing 101 (2019) 1431–1456.

[32] K. Dimolikas, A. V. Zarras, P. Vassiliadis, A study on the effect of a table's involvement in foreign keys to its schema evolution, in: 39th International Conference on Conceptual Modeling, ER 2020, Vienna, Austria, November 3-6, 2020, publisher = Springer, series = Lecture Notes in Computer Science, volume = 12400, pages = 456–470,, 2020.

[33] D. Braininger, W. Mauerer, S. Scherzinger, Replicability and reproducibility of a schema evolution study in embedded databases, in: ER 2020 Workshops, Vienna, Austria, November 3-6, 2020, volume 12584 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 210–219.

[34] P. Vassiliadis, 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, 2021, pp. 1–12.
[35] P. Vassiliadis, G. Kalampokis, Taxa and super taxa of schema evolution and their relationship to activity, heartbeat and duration, Inf. Syst. 110 (2022) 102109. URL: https://doi.org/10.1016/j.is.2022.102109. doi:10.1016/j.is.2022.102109.
[36] M. Klettke, H. Awolin, U. Störl, D. Müller, S. Scherzinger, 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, 2017, pp. 2462–2471.
[37] S. Scherzinger, S. Sidortschuck, An empirical study on the design and evolution of nosql database schemas, in: 39th International Conference on Conceptual Modeling, ER 2020, Vienna, Austria, November 3-6, 2020, volume 12400 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 441–455.
[38] U. Störl, M. Klettke, S. Scherzinger, 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, OpenProceedings.org, 2020, pp. 655–658.
[39] T. A. Limoncelli, SQL is no excuse to avoid devops, Commun. ACM 62 (2019) 46–49. URL: https://doi.org/10.1145/3287299. doi:10.1145/3287299.
[40] M. Stonebraker, R. C. Fernandez, D. Deng, M. L. Brodie, Database decay and what to do about it, Commun. ACM 60 (2017) 11. doi:10.1145/3014349.
[41] A. Maule, W. Emmerich, D. S. Rosenblum, Impact analysis of database schema changes, in: 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008, ACM, 2008, pp. 451–460.
[42] S. K. Gardikiotis, N. Malevris, A two-folded impact analysis of schema changes on database applications, Int. J. Autom. Comput. 6 (2009) 109–123.
[43] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou, Hecataeus: Regulating schema evolution, in: ICDE, 2010, pp. 1181–1184.
[44] M. Hartung, J. F. Terwilliger, E. Rahm, Recent advances in schema and ontology evolution, in: Z. Bellahsene, A. Bonifati, E. Rahm (Eds.), Schema Matching and Mapping, Data-Centric Systems and Applications, Springer, 2011, pp. 149–190.
[45] P. Manousis, P. Vassiliadis, A. V. Zarras, G. Papastefanatos, Schema evolution for databases and data warehouses, in: 5th European Summer School on Business Intelligence , eBISS 2015, volume 253 of *Lecture Notes in Business Information Processing*, Springer, 2015, pp. 1–31.
[46] L. Caruccio, G. Polese, G. Tortora, Synchronization of queries and views upon schema evolutions: A survey, ACM Trans. Database Syst. 41 (2016) 9:1–9:41.
[47] M. Goeminne, A. Decan, T. Mens, Co-evolving code-related and database-related changes in a data-intensive software system, in: IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, 2014, pp. 353–357.
[48] S. Scherzinger, W. Mauerer, H. Kondylakis, 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, 2021, pp. 2697–2700.
[49] P. Vassiliadis, F. Shehaj, G. Kalampokis, A. V. Zarras, Joint source and schema evolution: Insights from a study of 195 FOSS projects, in: Proceedings 26th International Conference

on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023, 2023, pp. 27–39. URL: https://doi.org/10.48786/edbt.2023.03. doi:10.48786/edbt.2023.03.

[50] S. A. Bohner, D. Gracanin, T. Henry, K. Matkovic, Evolutional insights from UML and source code versions using information visualization and visual analysis, in: Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2007, Banff, Alberta, Canada, June 25-26, 2007, 2007, pp. 145–148. doi:10.1109/VISSOF.2007.4290713.

[51] M. Krstajic, E. Bertini, D. A. Keim, Cloudlines: Compact display of event episodes in multiple time-series, IEEE Trans. Vis. Comput. Graph. 17 (2011) 2432–2439. URL: https://doi.org/10.1109/TVCG.2011.179. doi:10.1109/TVCG.2011.179.

[52] A. Kuhn, M. Stocker, Codetimeline: Storytelling with versioning data, in: 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland, 2012, pp. 1333–1336. doi:10.1109/ICSE.2012.6227086.

[53] O. Benomar, H. A. Sahraoui, P. Poulin, Visualizing software dynamicities with heat maps, in: 2013 First IEEE Working Conference on Software Visualization (VISSOFT), Eindhoven, The Netherlands, 2013, pp. 1–10. doi:10.1109/VISSOFT.2013.6650524.

[54] S. Rufiange, G. Melançon, Animatrix: A matrix-based visualization of software evolution, in: Second IEEE Working Conference on Software Visualization, VISSOFT 2014, Victoria, BC, Canada, September 29-30, 2014, 2014, pp. 137–146. doi:10.1109/VISSOFT.2014.30.

[55] B. Ens, D. J. Rea, R. Shpaner, H. Hemmati, J. E. Young, P. Irani, Chronotwigger: A visual analytics tool for understanding source and test co-evolution, in: Second IEEE Working Conference on Software Visualization, VISSOFT 2014, Victoria, BC, Canada, September 29-30, 2014, 2014, pp. 117–126. doi:10.1109/VISSOFT.2014.28.

[56] M. D. Feist, E. A. Santos, I. Watts, A. Hindle, Visualizing project evolution through abstract syntax tree analysis, in: 2016 IEEE Working Conference on Software Visualization, VISSOFT 2016, Raleigh, NC, USA, October 3-4, 2016, 2016, pp. 11–20. doi:10.1109/VISSOFT.2016.6.

[57] C. Mesnage, M. Lanza, White coats: Web-visualization of evolving software in 3d, in: S. Ducasse, M. Lanza, A. Marcus, J. I. Maletic, M. D. Storey (Eds.), Proceedings of the 3rd International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2005, Budapest, Hungary, September 25, 2005, IEEE Computer Society, 2005, pp. 40–45. URL: https://doi.org/10.1109/VISSOF.2005.1684302. doi:10.1109/VISSOF.2005.1684302.

[58] L. Meurice, A. Cleve, DAHLIA 2.0: A visual analyzer of database usage in dynamic and heterogeneous systems, in: 2016 IEEE Working Conference on Software Visualization, VISSOFT 2016, Raleigh, NC, USA, October 3-4, 2016, 2016, pp. 76–80. doi:10.1109/VISSOFT.2016.15.

[59] T. Schneider, Y. Tymchuk, R. Salgado, A. Bergel, Cuboidmatrix: Exploring dynamic structural connections in software components using space-time cube, in: 2016 IEEE Working Conference on Software Visualization, VISSOFT 2016, Raleigh, NC, USA, October 3-4, 2016, 2016, pp. 116–125. doi:10.1109/VISSOFT.2016.17.

[60] F. Pfahler, R. Minelli, C. Nagy, M. Lanza, Visualizing evolving software cities, in: Working Conference on Software Visualization, VISSOFT 2020, Adelaide, Australia, September 28 - October 2, 2020, IEEE, 2020, pp. 22–26. URL: https://doi.org/10.1109/VISSOFT51673.2020.00007. doi:10.1109/VISSOFT51673.2020.00007.