Information Systems 53 (2015) 363–385

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Growing up with stability: How open-source relational databases evolve

Ioannis Skoulis[a,1], Panos Vassiliadis[b,*], Apostolos V. Zarras[b]

[a] Opera, Helsinki, Finland
[b] University of Ioannina, Ioannina, Greece

A R T I C L E   I N F O

Available online 30 April 2015

Keywords:
Schema evolution
Software evolution
Lehman's laws

A B S T R A C T

Like all software systems, databases are subject to evolution as time passes. The impact of this evolution can be vast as a change to the schema of a database can affect the syntactic correctness and the semantic validity of all the surrounding applications. In this paper, we have performed a thorough, large-scale study on the evolution of databases that are part of larger open source projects, publicly available through open source repositories. Lehman's laws of software evolution, a well-established set of observations on how the typical software systems evolve (matured during the last forty years), has served as our guide towards providing insights on the mechanisms that govern schema evolution. Much like software systems, we found that schemata expand over time, under a stabilization mechanism that constraints uncontrolled expansion with perfective maintenance. At the same time, unlike typical software systems, the growth is typically low, with long periods of calmness interrupted by bursts of maintenance and a surprising lack of complexity increase.

*A truly stable system expects the unexpected, is prepared to be disrupted, waits to be transformed.*Tom Robbins, Even Cowgirls Get the Blues

1. Introduction

Software evolution is the change of a software system over time, typically performed via a remarkably difficult, complicated and time consuming process, software maintenance. Schema evolution is the most important aspect of software evolution that pertains to databases, as it can have a tremendous impact to the entire information system built around the evolving database, severely affecting both developers and end-users. Quite frequently, development waits till a "schema backbone" is stable and applications are build on top of it. This is due to the "*dependency magnet*" nature of databases: a change in the schema of a database may immediately drive surrounding applications to crash (in case of deletions or renamings) or be semantically defective or inaccurate (in the case of information addition, or restructuring). Therefore, discovering laws, patterns and regularities in schema evolution can result in great benefits, as we would be able to design databases with a view to their evolution and minimize the impact of evolution to the surrounding applications: (a) by avoiding "design anti-patterns" leading to cumulative complexity for both the database and the surrounding applications and (b) by planning administration and maintenance tasks and resources, instead of just responding to emergencies.

In sharp distinction to traditional software systems, and disproportionately to the severity of its implications, database evolution has hardly been studied throughout the entire lifetime of the data management discipline. It is only amazing

* Corresponding author.
E-mail addresses: giskou@gmail.com (I. Skoulis), pvassil@cs.uoi.gr (P. Vassiliadis), zarras@cs.uoi.gr (A.V. Zarras).
[1] Work conducted while in the University of Ioannina.

http://dx.doi.org/10.1016/j.is.2015.03.009
0306-4379/© 2015 Elsevier Ltd. All rights reserved.

to find out that, in the history of the discipline, just a handful of studies had been published in the area. The deficit is really amazing in the case of traditional database environments, where only two(!) studies [1,2] have been published. Apart from amazing, this deficit should also be expected: allowing the monitoring, study and eventual publication of the evolution properties of a database would expose the internals of a critical part of the core of an organization's information system. Fortunately, the open-source movement has provided us with the possibility to slightly change this landscape. As public repositories (git, svn, etc.) keep the entire history of revisions of software projects, including the schema files of any database internally hosted within them, we are now presented with the opportunity to study the version histories of such open source databases. Hence, within only a few years in the late '00's, several research efforts [3–6] have studied of schema evolution in open source environments. Those studies, however, focus on the statistical properties of the evolution and do not provide details on the mechanism that governs the evolution of database schemata.

To contribute towards amending this deficit, *the research goal of this paper involves the identification of patterns and regularities of schema evolution that can help us understand the underlying mechanism that governs it*. To this end, we study the evolution of the logical schema of eight databases, that are parts of publicly available, open-source software projects (Section 3). We have collected and cleansed the available versions of the database schemata for the eight case studies, extracted the changes that have been performed in these versions and, finally, we have come up with usable datasets that we subsequently analyzed.

Our main tool for this analysis came from the area of software engineering. In an attempt to understand the mechanics behind the evolution of software and facilitate a smoother, lest disruptive maintenance process, Meir Lehman and his colleagues introduced a set of rules in mid seventies [7], also known as the *Laws on Software Evolution* (Section 2). Their findings, that were reviewed and enhanced for nearly 40 years [8,9], have, since then, given an insight to managers, software developers and researchers, as to *what* evolves in the lifetime of a software system, and *why* it does so. Other studies (see [10] for a survey) have complemented these insights in this field, typically with particular focus to open-source software projects. In our case, we adapted the laws of software evolution to the case of schema evolution and utilized them as a driver towards understanding how the studied schemata evolve. Our findings (Section 4) indicate that the schemata of open source databases expand over time, with long periods of calmness connected via bursts of maintenance effort focused in time, and with significant effort towards the perfective maintenance of the schema that appears to result in an unexpected lack of complexity increase. Incremental growth of the schema is typically low and its volume follows a Zipfian distribution. In both the presentations of our results and in our concluding notes (Section 5) we also demonstrate that although the technical assessment of Lehman's laws shows that the typical software systems evolve quite differently than database schemata, the essence of the laws is preserved: evolution is not about uncontrolled expansion; on the contrary, there appears to be a stabilization mechanism that employs perfective maintenance to control the otherwise growing trend of increase in the information capacity of the database.

Roadmap: In Section 2, we summarize Lehman's laws for the non-expert reader and survey related efforts, too. In Section 3 we discuss the experimental setup of this study and in Section 4, we detail our findings. We conclude our deliberations with a summary of our findings and their implications in Section 5.

2. Lehman laws of software evolution in a nutshell

Meir M. Lehman and his colleagues, have introduced, and subsequently amended, enriched, and corrected a set of rules on the behavior of software as it evolves over time [7–9]. Lehman's laws focus on *E-type systems* that concern "software solving a problem or addressing an application in the real-world" [8]. The main idea behind the laws of evolution for E-type software systems is that their *evolution is a process that follows the behavior of a feedback-based system*. Being a feedback-based system, the evolution process has to balance (a) *positive feedback*, i.e., the need to adapt to a changing environment and grow to address the need for more functionality, and, (b) *negative feedback*, i.e., the need to control, constrain and direct change in ways that prevent the deterioration of the maintainability and manageability of the software. In the sequel, we list the definitions of the laws as they are presented in [9], in a more abstract form than previous versions and with the benefit of retrospect, after thirty years of maturity and research findings.

(I) *Law of Continuing Change*: An E-type system must be continually adapted or else it becomes progressively less satisfactory in use.

(II) *Law of Increasing Complexity*: As an E-type system is changed its complexity increases and becomes more difficult to evolve unless work is done to maintain or reduce the complexity.

(III) *Law of Self-regulation*: Global E-type system evolution is feedback regulated.

(IV) *Law of Conservation of Organizational Stability*: The work rate of an organization evolving an E-type software system tends to be constant over the operational lifetime of that system or phases of that lifetime.

(V) *Law of Conservation of Familiarity*: In general, the incremental growth (growth ratio trend) of E-type systems is constrained by the need to maintain familiarity.

(VI) *Law of Continuing Growth*: The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime.

(VII) *Law of Declining Quality*: Unless rigorously adapted and evolved to take into account changes in the operational environment, the quality of an E-type system will appear to be declining.

(VIII) *Law of Feedback System*: E-type evolution processes are multi-level, multi-loop, multi-agent feedback systems.

Before proceeding with our study, we present a first apodosis of the laws, taking into consideration both the wording of the laws, but most importantly their accompanying explanations [9].

An E-Type software system continuously changes over time (I) obeying a complex feedback-based evolution process (VIII). On the one hand, due to the need for growth and adaptation that acts as positive feedback, this process results in an increasing functional capacity of the system (VI), produced by a growth ratio that is slowly declining in the long term (V). The process is typically guided by a pattern of growth that demonstrates its self-regulating nature: growth advances smoothly; still, whenever there are excessive deviations from the typical, baseline rate of growth (either in a single release, or accumulated over time), the evolution process obeys the need for calibrating releases of perfective maintenance, i.e., code restructuring and documentation for better maintainability and comprehension (expressed via minor growth and demonstrating negative feedback) to stop the unordered growth of the system's complexity (III). On the other hand, to regulate the ever-increasing growth, there is negative feedback in the system controlling both the overall quality of the system (VII), with particular emphasis to its internal quality (II). The effort consumed for the above process is typically constant over phases, with the phases disrupted with bursts of effort from time to time (IV).

2.1. Lehman's laws and related empirical studies

Software evolution is an active research field for more than 40 years and concerns different levels of abstraction, including the software architecture [11], design [12] and implementation [10]. Lehman's theory of software evolution is the cornerstone of the efforts that have been performed all these years. For a detailed historical survey on the evolution of Lehman's theory and other related works the interested reader can refer to [10]. Following, we briefly discuss the milestones and key findings that resulted from these efforts.

Lehman's theory of software evolution was first introduced in the 70s. Back then, the theory included the first three laws, concerning the continuous change, the increasing complexity and the self-regulating properties of the software evolution process [7]. The experimental evidence that produced these laws was based on a single case study, namely the OS/360 operating system. During the 70s and the 80s the formulation of the first three laws has been revised, with respect to further results and empirical observations that came up [13]. Moreover, Lehman's theory has been extended with the fourth and the fifth law that concerned the issues of organizational stability and conservation of familiarity [13]. In the 90s, based on additional case studies, the laws have been revised again and extended with the last three laws, referring to the continuous growth, the declining quality and to the feedback mechanism that governs the evolution process [14,8,15]. Lehman's theory did not grow since then, the set of laws has been stabilized, and most of the activity around them concerned moderate changes in their formulation, performed in the 00s [9].

During all these years there have also been studies by other authors on the validity of the laws [16,17]. An interesting finding uncovered from these efforts is that the behavior of commercial software differs from that of academic and research software, with the former kind being much more faithful to the laws, compared to the latter two kinds. The partial validity of the laws is also highlighted in [18], along with the need for a more formal framework that would facilitate the assessment of the laws.

The diverse behavior of software concerning the validity of Lehman's laws is emphasized in subsequent studies that investigated the evolution of open source software. Most of these studies found only partial support for the validity of the laws. The efforts in this line of research vary from the pioneer studies of Godfrey and Tu [19,20], focusing mainly on Linux, to large scale studies [21–24]. The common ground in all these studies is that they found support for the laws of continuing change and growth. Refs. [23,25] concluded in the validation of more laws, including the ones of self-regulation and conservation of familiarity. Moreover, [26] revealed that the laws may be valid after a certain point in the software lifecycle. In particular, taking a step further from the efforts of Godfrey and Tu, [26] found that after a certain version the evolution of Linux follows, at least partially, most of the laws.

2.2. Empirical studies on database evolution

Being at the very core of most software, databases are also subject to evolution, which concerns changes in their contents and, most importantly, their schemas. Database evolution can concern (a) changes in the operational environment of the database, (b) changes in the content of the databases as time passes by, and (c) changes in the internal structure, or *schema*, of the database. *Schema evolution*, itself, can be addressed at (a) the conceptual level, where the understanding of the problem domain and its representation via an ER schema evolves, (b) the logical level, where the main constructs of the database structure evolve (for example, relations and views in the relational area, classes in the object-oriented database area, or (XML) elements in the XML/semi-structured area), and (c) the physical level, involving data placement and partitioning, indexing, compression, archiving, etc.

Interestingly, *the related literature on the actual mechanics of schema evolution includes only a few case studies*, as the research community would find it very hard to obtain access to monitor database schemata for an in depth study over a significant period of time. Despite the fact that in our work we study *schema evolution at the logical level of databases in open-source software*, here, we proceed to survey all the works we are aware about in the broader area of schema evolution.

The first paper [1] discusses the evolution of the database of a health management system over a period of 18 months, monitored by a tool specifically constructed for this purpose. A single database schema was examined, and the monitoring revealed that all the tables of the schema were affected and the schema had a 139% increase in size for relations and 274% for attributes. The consequences of this evolution were significantly large as a cumulative 45% of all the names that were used in the queries had to be deleted or inserted.

Fifteen years later, the authors of [3] made an analysis on the database back-end of MediaWiki, the software that powers Wikipedia. The study conducted over the versions of four years, revealed a 100% increase in schema size, the observation that around 45% of changes do not affect the information capacity of the schema (but are rather index adjustments, documentation, etc.), and a statistical study of lifetimes, change breakdown and version commits. Special mention should be made to this line of research [27], as it is based on PRISM (recently re-engineered to PRISM++ [28]), a change management tool that provides a language of Schema Modification Operations (SMO) (that model the creation, renaming and deletion of tables and attributes, and their merging and partitioning) to express schema changes. More importantly, the people involved in this line of research should be credited for providing a large collection of links[2] for open source projects that include database support.

A work in the area of data warehousing [2] monitored the evolution of seven ETL scripts along with the evolution of the source data. The experimental analysis of the authors is based in a six-month monitoring of seven real-world ETL scenarios that process data for statistical surveys. The findings of the study indicate that schema size and module complexity are important factors for the vulnerability of an ETL flow to changes. This work has been part of an effort to provide what-if analysis facilities to the management of schema evolution via the Hecataeus tool (see [29,30]).

Finally, certain efforts studied the evolution of databases, while taking into account the applications that use them. In particular, in [5] the authors considered 4 case studies of embedded databases (i.e., databases tightly coupled with corresponding applications that rely on them) and studied the different kinds of changes that occurred in these cases. Moreover, they performed a respective frequency and timing analysis, which showed that the database schemas tend to stabilize over time. In [4], the authors focused on two case studies. The results of this effort revealed that database schema changes and that the source code of dependent applications does not always evolve in sync with changes to the database schema. Ref. [4] further provides a discussion concerning that the impact of database schema changes on the application code. Ref. [6] takes a step further with an empirical study of the co-evolution of database schemas and code. This effort investigated ten case studies. The results indicate that database schemas evolve frequently during the application lifecycle, with schema changes implying a significant amount of code level modifications.

2.3. Novelty with respect to the state of the art

Going beyond the related literature on software evolution, in general, and database evolution, in particular, our CAiSE'14 paper [31] investigated for the first time patterns and regularities of database evolution, based on Lehman's laws. To this end, we conducted a large scale case study of eight databases, that are parts open-source software projects. This paper extends our prior work with further details concerning the intuition and the relevance of the laws in the case of databases, the metrics that have been used in the literature for the assessment of the laws, and the metrics that we employed in the case of databases. More importantly, we provide detailed presentations of the results and thorough discussions of our findings.

3. Experimental setup of the study

Datasets: We have studied eight database schemata from open-source software projects. Fig. 1 lists the datasets along with some interesting properties.

ATLAS[3] is a particle physics experiment at the Large Hadron Collider at CERN, Geneva, Switzerland, with the goal of learning about the basic forces that have shaped our universe. ATLAS Trigger is the software responsible for filtering the immense data (40 TB per second) collected by the Collider and storing them in its Oracle database.

BioSQL[4] is a generic relational model covering sequences, features, sequence and feature annotation, a reference taxonomy, and ontologies (or controlled vocabularies) from various sources such as GenBank or Swissport. While originally conceived as a local relational store for GenBank, the project has since become a collaboration platform between the Open Bioinformatics Foundation (OBF) projects (including BioPerl, BioPython, BioJava, and BioRuby). The goal is to build a sufficiently generic schema for persistent storage of sequences, features, and annotation in a way that is interoperable between these Bio* projects.

Ensembl is a joint scientific project between the European Bioinformatics Institute (EBI)[5] and the Wellcome Trust Sanger Institute (WTSI)[6] which was launched in 1999 in response to the imminent completion of the Human Genome Project. The goal of Ensembl was to automatically annotate the three billion base pairs of sequences of the genome, integrate this annotation with other available biological data and make all this publicly available via the web. Since the launch of the website, many more genomes have been added to Ensembl and the range of available data has also expanded to include comparative genomics, variation and regulatory data.

MediaWiki[7] was first introduced in early 2002 by the Wikimedia Foundation along with Wikipedia, and hosts Wikipedia's content since then. As an open source system (licensed under the GNU GPL) written in PHP, it was also adopted by many companies and is used in thousands of websites both as a knowledge management system, and for collaborative group projects.

[2] http://yellowstone.cs.ucla.edu/schema-evolution/index.php/Benchmark_Extension

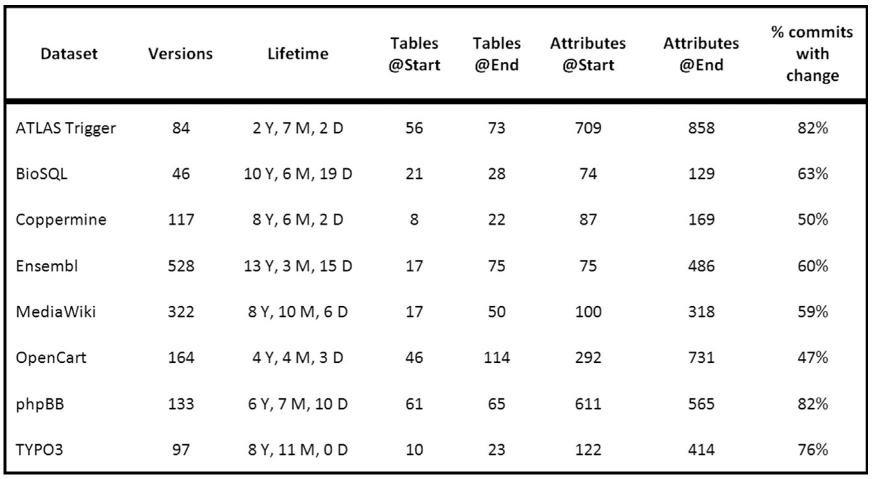
[3] http://atlas.web.cern.ch/Atlas/Collaboration/

[4] http://www.biosql.org/wiki/Main_Page

[5] https://www.ebi.ac.uk/

[6] https://www.sanger.ac.uk/

[7] https://www.mediawiki.org/wiki/MediaWiki



Dataset	Versions	Lifetime	Tables @Start	Tables @End	Attributes @Start	Attributes @End	% commits with change
ATLAS Trigger	84	2 Y, 7 M, 2 D	56	73	709	858	82%
BioSQL	46	10 Y, 6 M, 19 D	21	28	74	129	63%
Coppermine	117	8 Y, 6 M, 2 D	8	22	87	169	50%
Ensembl	528	13 Y, 3 M, 15 D	17	75	75	486	60%
MediaWiki	322	8 Y, 10 M, 6 D	17	50	100	318	59%
OpenCart	164	4 Y, 4 M, 3 D	46	114	292	731	47%
phpBB	133	6 Y, 7 M, 10 D	61	65	611	565	82%
TYPO3	97	8 Y, 11 M, 0 D	10	23	122	414	76%

Fig. 1. The datasets employed in our study.

Coppermine[8] is a photo gallery web application. OpenCart[9] is an open source shopping cart system. PhpBB[10] (PHP Bulletin Board) is an Internet forum package written in PHP. TYPO3[11] is a web content management framework based on PHP. All these platforms are highly rated and used.

Dataset Collection and Processing: A first collection of links to available datasets was made by the authors of [3,27][12]; for this, these authors deserve honorable credit. We isolated eight databases that appeared to be alive and used (as already mentioned, some of them are actually quite prominent). For each dataset, we have gathered the schema versions (DDL files) that were available at June 2013, directly from public source code repositories (cvs, svn, git) for the eight datasets listed in Fig. 1. We have targeted main development branches and trunks to maximize the validity of the gathered resources. *We are interested only on changes of the database part of the project as they are integrated in the trunk of the project*. Hence, we collected all the commits of the trunk or master branch that were available at the time, and ignored all other branches of the project, as well as any commits of other modules of the project that did not affect the database.

For all of the projects, we focused on their release for MySQL (except ATLAS Trigger, available only for Oracle). Those files were then renamed with their filenames matching to the date (in standard UNIX time) the commit was made. The files were then processed in sequential pairs from our tool, Hecate, to give us in a fully automated way (a) the differences between two subsequent commits and (b) the measures we needed to conduct this study. Attributes are marked as altered if they exist in both versions and their type or participation in their tables's primary key changed. Tables are marked as altered if they exist in both versions and their contents have changed (attributes inserted/deleted/altered).

[8] http://coppermine-gallery.net/
[9] http://www.opencart.com
[10] https://www.phpbb.com/
[11] http://typo3.org/
[12] http://data.schemaevolution.org

All the datasets used, along with our tool-suite for managing the evolution of databases can be found in our group's git: https://github.com/DAINTINESS-Group.

4. Assessing the laws for schema evolution

The laws of software evolution where developed and reshaped over forty years. Explaining each law in isolation from the others is precarious, as it risks losing the deeper essence and inter-dependencies of the laws [9]. To this end, in this section, we organize the laws in three thematic areas of the overall evolution management mechanism that they reveal. The first group of laws discusses the existence of a feedback mechanism that constrains the uncontrolled evolution of software. The second group discusses the properties of the growth part of the system, i.e., the part of the evolution mechanism that accounts for positive feedback. The third group of laws discusses the properties of perfective maintenance that constrains the uncontrolled growth, i.e., the part of the evolution mechanism that accounts for negative feedback. To quantitatively support our study, we utilize the following measures:

- *Schema size of a version*: The number of tables of a schema version.
- *Schema Growth*: The difference between the schema size of two (typically subsequent) versions (i.e., new–old).
- *Heartbeat*: A sequence of tuples, one per transition, with the count of the events that occurred during this transition. In the context of this paper, for each transition between two subsequent versions, we produce a tuple of measures including Table Insertions, Table Deletions, Attribute Insertions, Attribute Deletions, Attribute Alternations (change of data type), Attributes Inserted at Table Formation, Attribute Deletions at Table Removal. To clarify, Attribute Insertions concern additions of attributes to an existing table, whereas Attributes Inserted at Table Formation concern the number of attributes generated whenever a new table

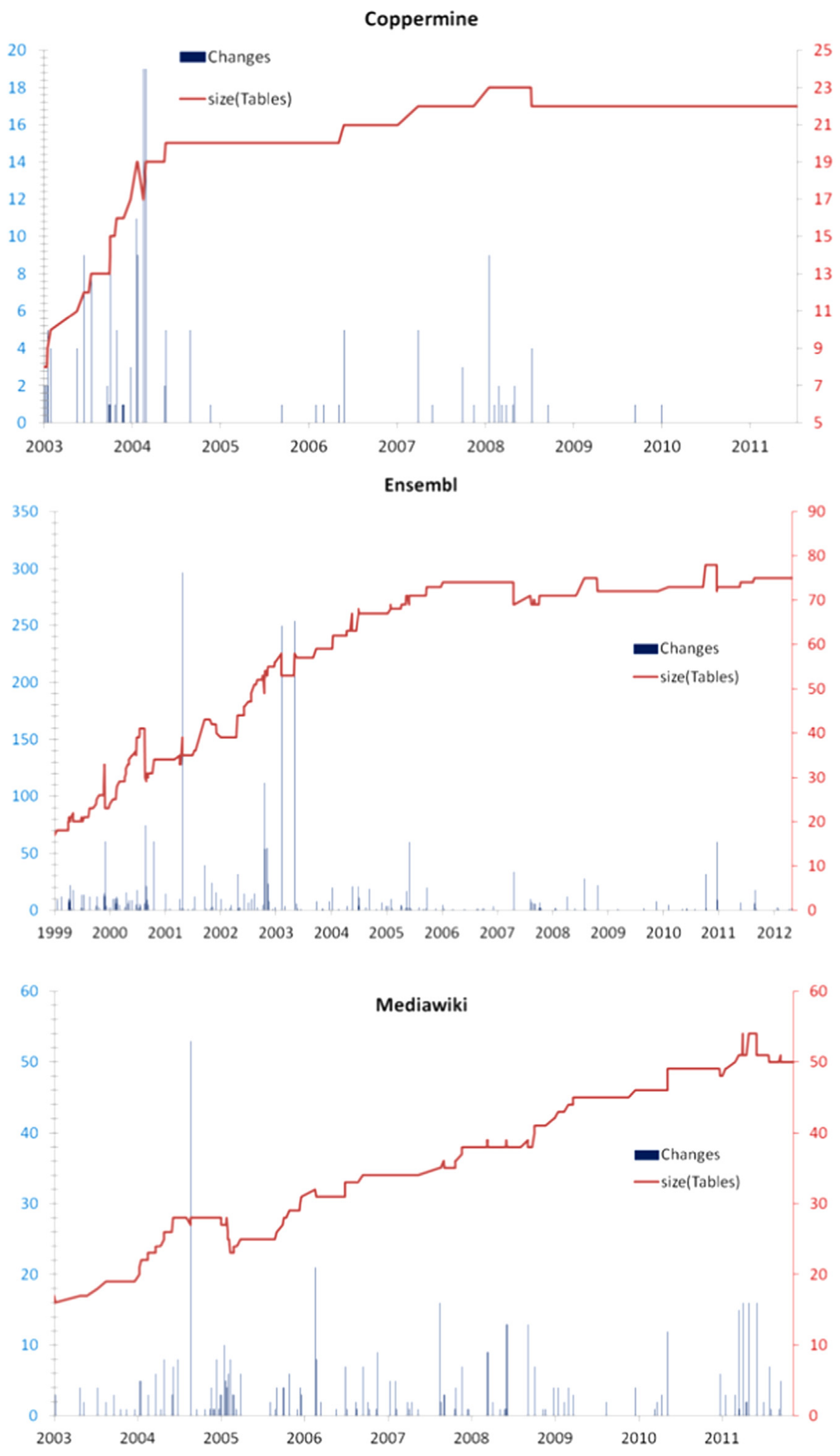


Fig. 2. Combined demonstration of heartbeat (number of changes per version) and schema size (no. of tables). The left axis signifies the amount of change and the right axis the number of tables.

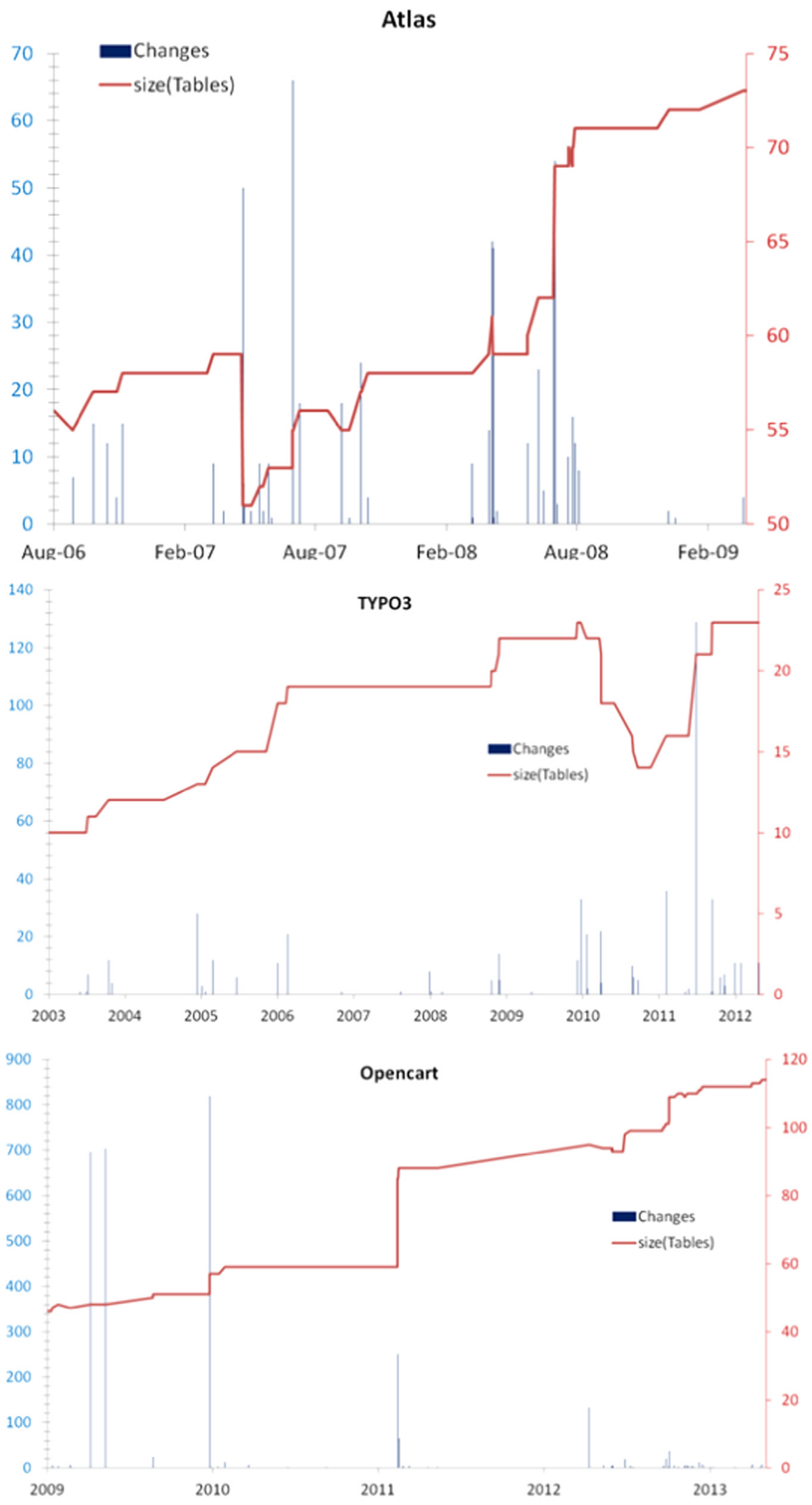


Fig. 3. Combined demonstration of heartbeat (continued).

is born. Attribute Deletions concern deletions from a table that continues to exist, whereas Attribute Deletions at Table Removal concern attributes that are removed whenever their containing table is removed. We sum up these measures per transition, to produce the Heartbeat of the lifetime of the dataset.

We would like to remind the reader that we study the *evolution of the logical schema of databases in open-source software*. In all our deliberations, we take the above context as granted and avoid repeating it for reasons of better presentation of our results.

4.1. Is there a feedback-based system for schema evolution?

4.1.1. Law of continuing change (Law I)

The first law argues that the system continuously changes over time.

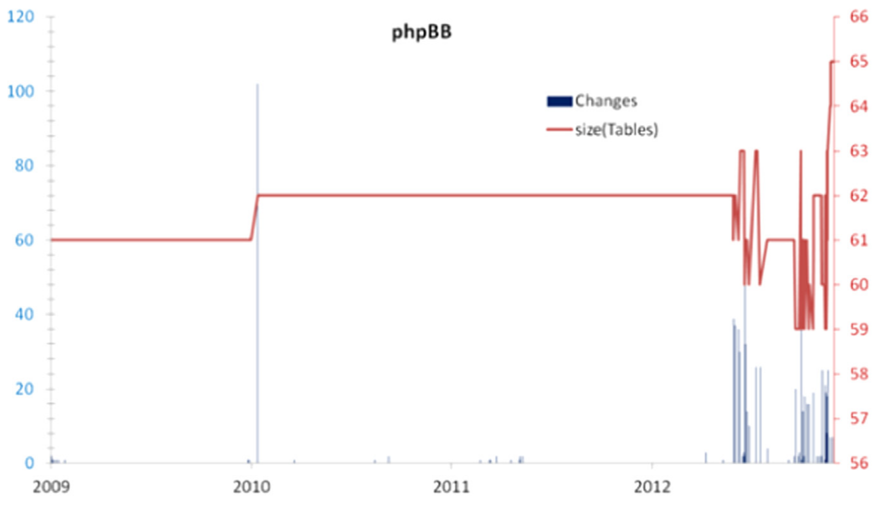
An E-type system must be continually adapted or else it becomes progressively less satisfactory in use.

The main idea behind this law is simple: as the real world environment evolves, the software that is intended to address its problems has to evolve too. If this does not happen, the system becomes less satisfactory.

Metrics for the assessment of the law's validity: To establish the law, one needs to show that the software shows signs of evolution as time passes. Possible metrics from the field of software engineering [23] include (a) the cumulative number of changes and (b) the breakdown of changes over time.

Assessment: To validate the hypothesis that the law of continuing change holds, we study the heartbeat of the schema's life (see Figs. 2–4 for a combined demonstration of heartbeat and schema size).

With the exception of BioSQL that appeared to be "sleeping" for some years and was later re-activated, in all other cases, we have changes (sometimes moderate, sometimes



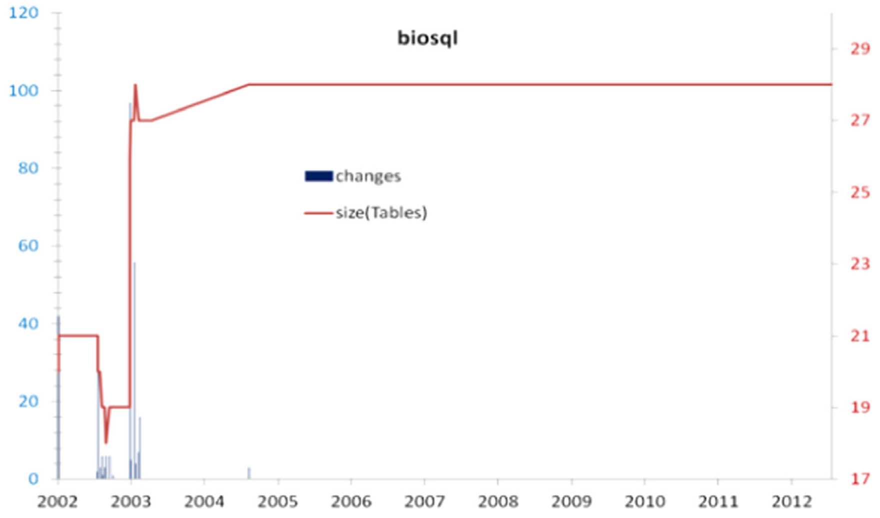


Fig. 4. Combined demonstration of heartbeat (continued).

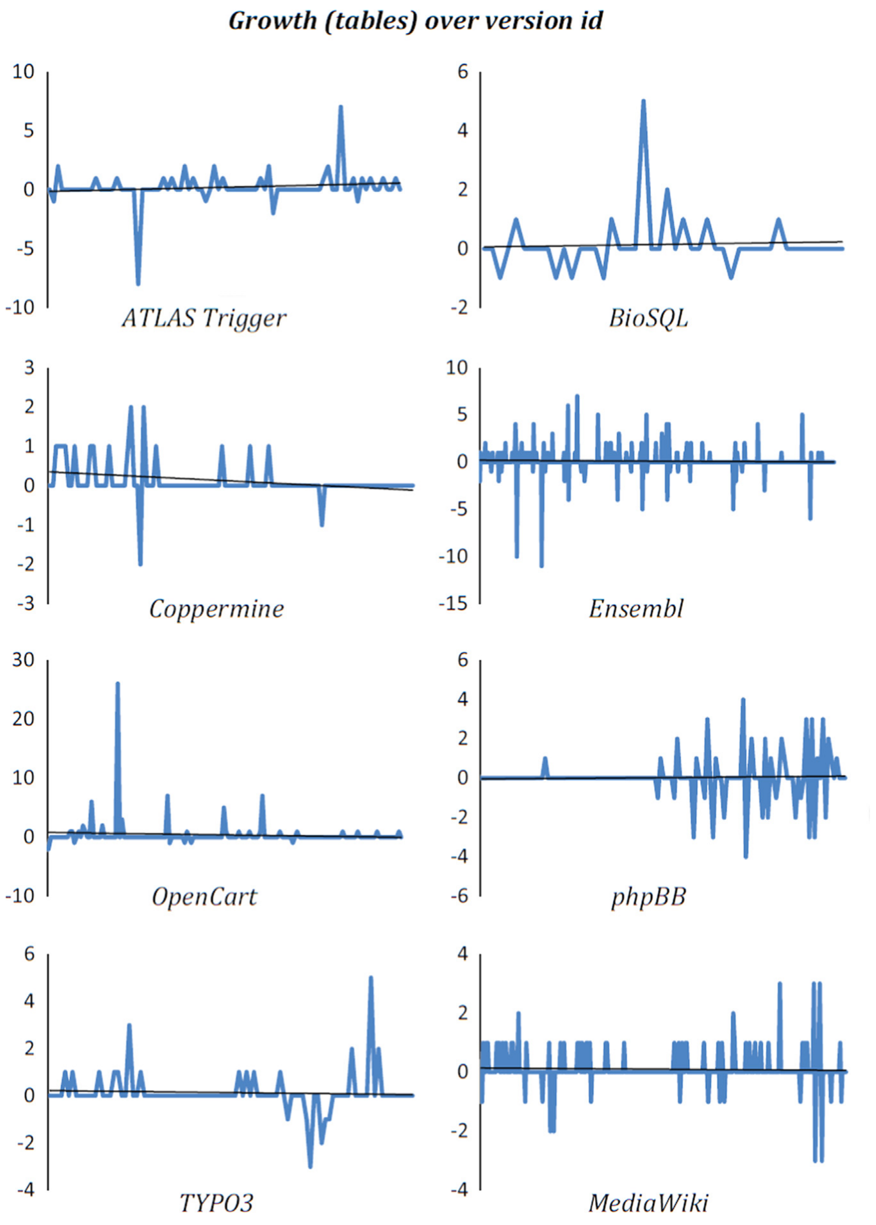


Fig. 5. Growth (tables) over version id for all the datasets.

even excessive) over the entire lifetime of the database schema. An important observation stemming from the visual inspection of our change-over-time data, is that the term *continually* in the law's definition is challenged: *we observe that database schema evolution happens in bursts, in grouped periods of evolutionary activity, and not as a continuous process*! Take into account that the versions with zero changes are versions where either commenting and beautification takes place, or the changes do not refer to the information capacity of the schema (relations, attributes and constraints) but rather, they concern the physical level properties (indexes, storage engines, etc.) that pertain to performance aspects of the database.

Can we state that this stillness makes the schema “unsatisfactory” (referring back to the wording of the first law by Lehman)? We believe that the answer to the question is negative: since the system hosting the database continues to be in use, user dissatisfaction would actually call for continuous growth of the database, or eventual rejection of the system. This does not happen. On the other hand, our explanation relies on the reference nature of the database in terms of software architecture: if the database evolves, the rest of the code, which is basically using the database (and not vice versa), breaks.

Overall, if we account for the exact wording of the law, we conclude that the law partially holds.

4.1.2. Law of Self-regulation (Law III)

The third law of software evolution is known as the law of “Self-regulation” and comes with a laconic definition.

Global E-type system evolution is feedback regulated.

The main idea behind this law is that the system under development is actually a feedback-regulated system: development and maintenance take place and there is positive and negative feedback to the system. As the clients of the system request more functionality, the system grows in size to address this demand; at the same time, as the system grows, corrective and perfective maintenance has to take place to remove bugs and improve the internal quality of the software (reduced complexity, increased understandability) [8].

Thus, the system's growth cannot continually evolve with the same rate; on the contrary, what one expects is to see a typical "baseline" growth, interrupted with releases of perfective maintenance. This trend is so strong, that, in the long run, the system's size demonstrates what the authors of [8] call "cyclic effects" and the authors of [9] call "patterns of growth".

Metrics for the assessment of the law's validity: Whereas the law simply states that the evolution of software is feedback regulated, its experimental validation in the area of software systems is typically supported by the observation of a recurring pattern of smooth expansion of the system's size that is interrupted with releases with size reductions or abrupt growth. Moreover, due to a previous wording of the law (e.g., see [8]) that described change to follow a normal distribution, the experimental assessment included the validation of whether growth demonstrates oscillations around an average value [7–9]. The ripples in the size of the system are assumed to indicate the existence of feedback in the system: positive feedback results in the system's expansion and negative feedback involves perfective maintenance coming with reduced rate of growth (which is not due to functional growth but re-engineering towards better code quality) – if not with system shrinking (due to removal of unnecessary parts or their merging with other parts).

Assessment: We organize the discussion of our findings around size and growth, both of which demonstrate some patterns, although not the ones expected by the previous literature.

Size: The evolution of size can be observed in Figs. 2–4. Concerning the issue of a recurring, fundamental pattern of smooth expansion, interrupted with abrupt changes or, more generally, versions of perfective maintenance, we have to say that we simply cannot detect the behavior that Lehman did (contrast Figs. 2–4 to the respective figures of articles [7,8]): in sharp contrast to the smooth *baseline* growth that Lehman has highlighted, the evolution of the size of the studied database schemata provides a landscape with a large variety of *sequences of the following three fundamental behaviors*.

- In all schemata, we can see *periods of increase*, especially at the beginning of their lifetime or after a large drop in the schema size. This is an indication of positive feedback, i.e., the need to expand the schema to cover the information needs of the users – especially since the overall trend in almost all of the studied databases is to see an increase in the schema size as time passes.
- In all schemata, there are *versions with drops in schema size*. Those drops are typically sudden and steep and usually take place in short periods of time. Sometimes, in fact, these drops are of significantly larger size than the typical change. We can safely say that the existence of these drops in the schema size indicates perfective maintenance and thus, the existence of a negative feedback mechanism in the evolution process.
- In all schemata, there are *periods of calmness*, i.e., periods of non-modification to the logical structure of the schema. This is especially evident if one observes the heartbeat of the database, where changes are grouped to very specific moments in time.

Growth and its oscillations: Growth (i.e., the difference in the size between two subsequent versions) comes with common characteristics in all datasets. In most cases, growth is small (typically ranging within 0 and 1). As Fig. 5 demonstrates, we *have too many occurrences of zero growth*, typically iterating between small non-zero growth and zero growth. Due to perfective maintenance, we also have negative values of growth (less than the positive ones). We do not have a constant flow of versions where the schema size is continuously changing; rather, we have small spikes between one and zero. Thus, we have to state that the growth comes with *a pattern of spikes*. Due to this characteristic, *the average value is typically very close to zero* (*on the positive side*) *in all datasets*, *both for tables and attributes*. There are *few cases of large change* too; we forward the reader to Law V for a discussion and to Fig. 9 for a graphical depiction of their characteristics.

The oscillations of growth demonstrates other patterns too: it is quite frequent, especially at the attribute level, to see *sequences of oscillations of large size*: i.e., an excessive positive delta followed immediately by an excessive negative growth (see Fig. 9). We do, however, observe the oscillations between positive and negative values (remember, the average value is very close to zero), much more on the positive side, however, with several occasions of excessive negative growth (clearly demonstrating perfective maintenance).

We would like to put special emphasis to the observation that *change is small* . In terms of tables, growth is mostly bounded in small values. This is not directly obvious in the charts, because they show the ripples; however, almost all numbers are in the range of $[-2..2]$ – in fact, mostly in the range $[0..2]$. Few abrupt changes occur. In terms of attributes (Fig. 6), the numbers are higher, of course, and depend on the dataset. Typically, those values are bounded within $[-20,20]$. However, the deviations from this range are not many.

In the course of our deliberations, we have observed a pattern common in all datasets: *there is a Zipfian model in the distribution of frequencies*. Observe Fig. 7 that comes with two parts, both depicting how often a growth value appears in the attributes of Ensemble. The *x*-axis keeps the delta size and the *y*-axis the number of occurrences of this delta. In the upper part we include zeros in the counting (343 occurrences out of 528 data points) and in the lower part we exclude them (to show that the power law does not hold only for the most popular value). We observe that there is a small range of deltas, between -2 and 4 that takes up 450 changes out of the 528. This means that, despite the large outliers, change is strongly biased towards small values close to zero.

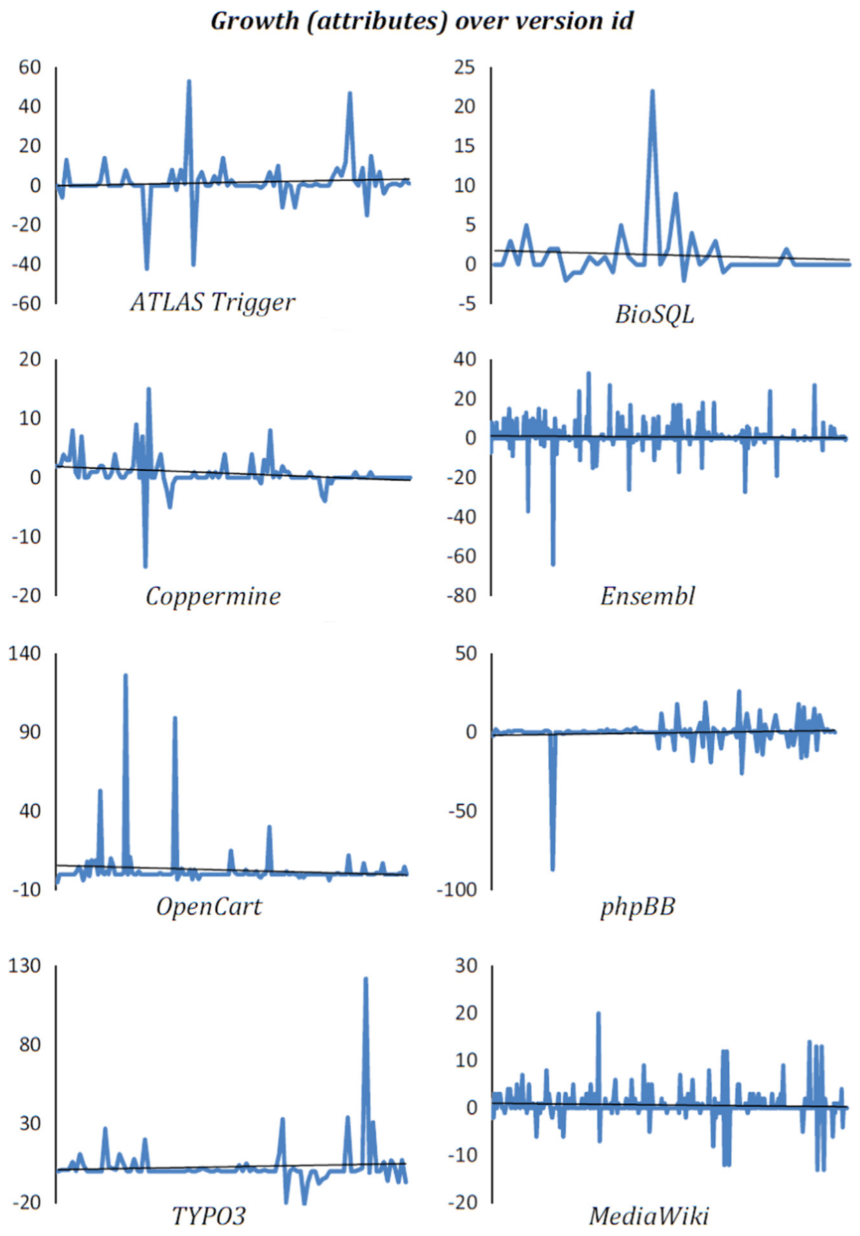


Fig. 6. Growth (attributes) over version id for all the datasets; we measure attribute growth as the difference in the total number of attributes of all tables, between two subsequent versions.

In fact, both phenomena observed here, i.e., (a) the bounded small change around zero, (b) following a Zipfian distribution of frequencies, constitute two of the patterns that are global to all datasets and without any exceptions whatsoever.

Despite the fact that change does not follow the pattern of baseline smooth growth of Lehman and the fact that change obeys a Zipfian distribution with a peak at zero, we believe that the presence of feedback in the evolution process is clear; thus the law holds.

4.1.3. *Law of Feedback System (Law VIII)*

The eighth law of software evolution is known as the law of “Feedback System”.

E-type evolution processes are multi-level, multi-loop, multi-agent feedback systems.

The main idea around this law refers to the fact that original “observation has shown that the system behaves as self-stabilizing feedback system” [14]. There is a big discussion in the literature on various components and actors whose interactions limit and guide the possible ways via which the system can evolve. We refer the interested reader to [9] for this. From our part, we do not presume to fully know the mechanics that constraint the growth of a database schema. However, we can focus to the part that there is indeed a mechanism that stabilizes the tendency for uninterrupted growth of the schema – and in fact we can try to assess

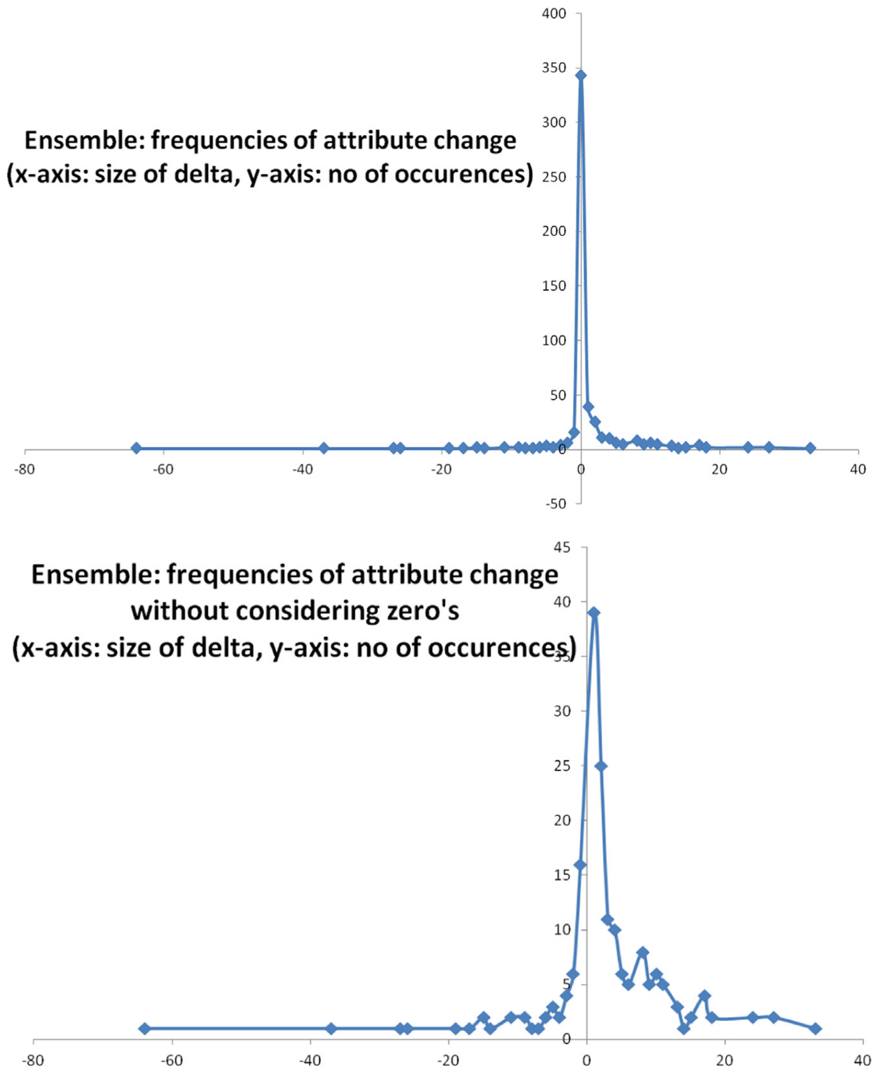


Fig. 7. Frequency of change values for Ensembl attributes.

whether this is a regressive mechanism whose behavior can be generally estimated.

Metrics for the assessment of the law's validity: To assume the law as valid we need to establish that it is possible to simulate the evolution of the schema size via an accurate formula. Following [8,15], we will perform *regression analysis* to *estimate* the number of relations for each version of the schema. We adopt the formulas found at [8,15] on the relationship of the new size of the system as a function of the previous size of it, adapted via an "*inverse square*" feedback effect. The respective formula is

$$\hat{S}_i = \hat{S}_{i-1} + \frac{\overline{E}}{\hat{S}_{i-1}^2}$$
$$\overline{E} = avg(E_i), \quad i = 1 \dots n \tag{1}$$

where $\hat{S}$ refers to the estimated system size and $\overline{E}$ is a model parameter approximating effort. Related literature [8] suggests computing $\overline{E}$ as the average value of individual E_i, one per transition. To estimate these individual effort approximations, E_i, the authors of [8] suggest two alternative formulae:

$$E_i = (s_i - s_{i-1}) \cdot s_{i-1}^2 \tag{2}$$

$$E_i = \frac{s_i - s_1}{\sum_{j=1}^{i-1} \frac{1}{s_j^2}} \tag{3}$$

Assessment: We now move on to discuss what seems to work and what not for the case of schema evolution. We will use the OpenCart dataset as a reference example; however, all datasets demonstrate exactly the same behavior.

The main challenge with formula (1) is the estimation of $\overline{E}$. As a first step, we have generalized the formulae (2) and (3) via a parameterized expression:

$$E_i = \frac{s_i - s_\alpha}{\sum_{j=\alpha}^{i-1} \frac{1}{s_j^2}} \tag{4}$$

where s_i refers to the actual size of the schema at version i and α refers to the version from which counting starts. The model of [8] comes with two values for α, specifically (i) $\alpha = i-1$ for formula (2), and (ii) $\alpha = 1$ for formula (3). The essence of the formula is that, to compute E_i, we use α previous versions to estimate effort.

Then we began our assessment. First, we assessed the formulae of [8]. In this case, we compute the average $\overline{E}$ of the individual E_i over the entire dataset. We employ four different values for α, specifically $i-1$ (last version), 1 (for the entire dataset) and 5 and 10 for the respective past versions. We depict the result in Fig. 8(top), where the actual size is represented by the blue solid line. The results indicate that the approximation modestly succeeds in predicting an overall increasing trend for all four cases, and, in fact, all four approximations targeted towards predicting an increasing tendency that the actual schema does not demonstrate. At the same time, all four approximations fail to capture the individual fluctuations within the schema lifetime.

Then, we tried to improve on this result, and instead of computing $\overline{E}$ as the total average over all the values of the dataset, we compute it as the running average (not assuming a global average, but tuning the average effort with every added release). In this case, depicted in Fig. 8 (middle), the results are less satisfactory than our first attempt.

After these attempts, we decided to alter the computation of $\overline{E}$ again. A better estimation occurred when we realized that back in 1997 people considered that the parameter $\overline{E}$ was constant over the entire lifetime of the project; however, later observations (see [9]) led to the revelation that the project was split into phases. So, for every version i, we compute $\overline{E}$ as an average over the last τ E_j values, with small values for τ (1/5/10) – contrast this to the previous two attempts where $\overline{E}$ was computed as a total average over the entire dataset (i.e., constant for all versions) or a running average from the beginning of the versions till the current one.

So, the main formula of the law is restated (and actually generalized), by replacing a global parameter $\overline{E}$ with a varying parameter $\overline{E}^i$ that can change per version (thus the superscript notation signifies the value of the effort estimation at version i). The versions used for this calculation are within the range $[\tau^s, \tau^e]$:

$$\widehat{S}_i = \widehat{S}_{i-1} + \frac{\overline{E}^i}{\widehat{S}^2_{i-1}}, \quad \overline{E}^i = avg_{j=\tau^s}^{\tau^e}(E_j) \tag{5}$$

For the three simulation attempts that we have run, we have the following configurations:

Method	Values for $[\tau^s, \tau^e]$	
Global average	τ^s: 1	τ^e: n
Running average	τ^s: 1	τ^e: $i-1$
Last τ vs. ($\tau \in \{1, 5, 10\}$)	τ^s: $i-\tau$	τ^e: $i-1$

We also decided to use the last 5 or 10 versions to compute E_i, i.e., α is 5 or 10. This has already been used in the past experiments too.

As we can see in Fig. 8(bottom), *the idea of computing the average* $\overline{E}$ *with a short memory of* 5 *or* 10 *versions produced extremely accurate results. This holds for all datasets*. This observation also suggests that, if the phases that [9] mentioned actually exist for the case of database schema, they are really small, or non-existent, and a memory of 5–10 versions is enough to produce very accurate results. The fact that this works with $\tau = 1$, and in fact, better than the other approximations is puzzling and counters the existence of phases.

We do not have a convincing theory as to why the formula works. We understand that there are no constants in the feedback system and in fact, the feedback mechanism needs a second feedback loop, with a short memory for estimating the model parameter $\overline{E}$. In plain words, this signifies that both size and effort approximation are intertwined in a multi-level feedback mechanism.

Overall, *the evolution of the database schema appears to obey the behavior of a feedback-based mechanism, as the schema size of a certain version of the database can be accurately estimated via a regressive formula that exploits the amount of changes in recent, previous versions*.

4.2. Properties of growth for schema evolution

Growth occurs as *positive feedback* to the system, in an attempt to expand the system with more functionality, or address new assumptions that make its operation acceptable, e.g., new user requirements, and an evolving operational environment. In this subsection, we study the properties of the growth.

4.2.1. Law of Continuing Growth (Law VI)

The sixth law of software evolution is known as the law of "Continuing Growth".

> The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime.

The sixth law resembles the first law (continuing change) at a fist glance; however, as explained in [8], these two laws cover different phenomena. The first law refers to the necessity of a software system to adapt to a changing world. The sixth law refers to the fact that a system cannot include all the needed functionality in a single version; thus, due to non-elastic time and resource constraints, several desired functionalities of the system are excluded from a version. As time passes, these functionalities are progressively blended in the system, along with the new requirements stemming from the first law's context of an evolving world. As [9] eloquently states "the former is primarily concerned with functional and behavioral change, whereas the latter leads, in general, directly to additions to the existing system and therefore to its growth".

Metrics for the assessment of the law's validity: Possible metrics for the sixth law that come from the software engineering community [23] include LOC, number of definitions (of types, functions and global variables) and number of modules. We express again a point of concern here: it is impossible to discern, from this kind of "black-

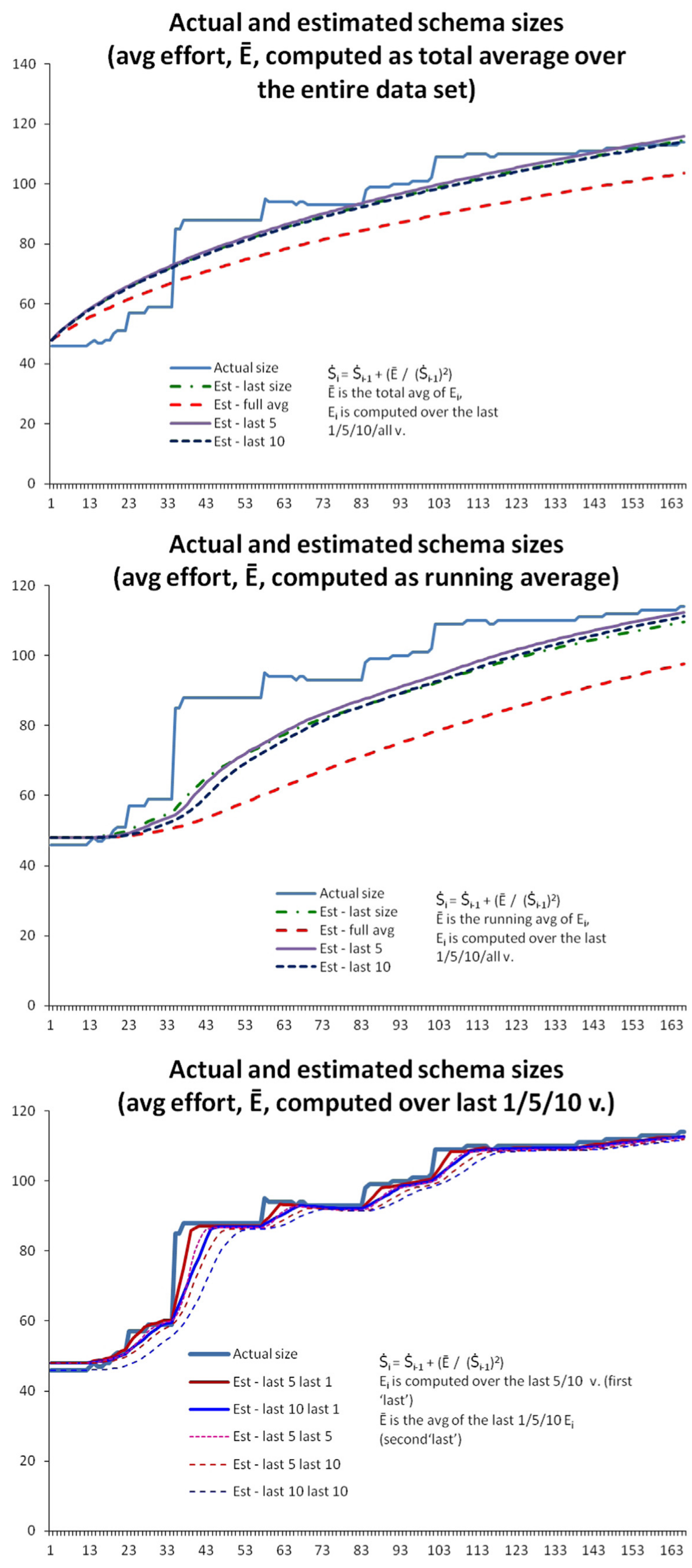


Fig. 8. Actual and estimated schema size for OpenCart via a total (top), running (middle) or bounded (bottom) averages of individual E_i. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

box" measurements, the percentage of change that pertains to the context of the law of continuing growth. Ideally, one should count the number of recorded "ToDo" functionalities blended within each version. However, we do recognize that this task is extremely hard to *automate at a large scale*. In our case, as we mainly refer to information capacity rather than physical level schema properties, we can utilize the schema size as a safe measure of observing "additions to the existing system".

Assessment: In all occasions, the schema size increases in the long run (Figs. 2–4). We frequently observe some shrinking events in the timeline of schema growth in all datasets. However, *all datasets demonstrate the tendency to grow over time*.

At the same time, we also have differences from traditional software systems: as with Law I, the term "continually" is questionable. As already mentioned (refer to Law III and Figs. 2–4), change comes with frequent (and sometimes long) periods of *calmness*, where the size of the schema does not change (or changes very little). Calmness is clearly a phenomenon not encountered in the study of traditional software systems by Lehman and acquires extra importance if one also considers that in our study we have isolated only the commits to the files with the database schema and not the commits to the entire information system that uses it: this means that there are versions of the system, for which the schema remained stable while the surrounding code changed.

Therefore we can conclude that *the law holds* (*the information capacity of the database schema is enhanced in the long run*), *albeit modified to accommodate the particularities of database schemata* (*changes are not continuous but rather, they come within large periods of calmness*).

4.2.2. *Law of Conservation of Familiarity (Law V)*

The fifth law of software evolution is known as the law of "Conservation of Familiarity".

> In general, the incremental growth (growth ratio trend) of E-type systems is constrained by the need to maintain familiarity.

As the system evolves, all the stakeholders that are associated to it (developers, users, managers, etc.) must spend effort to understand and actually, master its content and functionality. Whenever there is excessive growth in a version, the feedback mechanism tends to diminish the growth in subsequent versions, so that the change's contents are absorbed by people. Interestingly, whereas the original form of the law refers to a constant (statistically invariant) rate, the new version of the law is accompanied by explanations strongly indicating *a* "*long term decline in incremental growth and growth ratio … of all release-based systems studied*" [9]. This result came as experimental evidence from the observation of several systems, accompanied by the anecdotal evidence of a growing imbalance in volume in favor of corrective versus adaptive maintenance. Refs. [23,15] also give a corollary of the law stating that versions with high volume of changes are followed by versions performing corrective or perfective maintenance.

Metrics for the assessment of the law's validity: Ref. [9] gives a large list of possible metrics: objects, lines of code, modules, inputs and outputs, interconnections, subsystems, features, requirements, and so on. Ref. [23] proposes metrics that include (i) the growth of the system, (ii) the growth ratio of the system, and (iii) the number of changes performed in each version. We align with these tactics and use the schema growth of the involved datasets.

To validate the law we need to establish the following facts:

- The growth of the schema is not increasing over time; in fact, it is – at best – constant or, more realistically, it declines over time/version. A question, typically encountered in the literature, is: "What is the effect of age over the growth and the growth ratio of the schema?" Is it slowly declining, constant or oblivious to age? To address this question, we produce a linear interpolation of the growth per dataset to show its overall trend (Fig. 5).
- Another question of interest in the related literature is: "What happens after excessive changes? Do we observe small ripples of change, showing the absorbing of the change's impact in terms of corrective maintenance and developer acquaintance with the new version of the schema?" In this case, the pattern we can try to establish is that abrupt changes are followed by versions where developers absorb the impact of the change and produce minor modifications/corrections, thus resulting in versions with small growth following the version with significant difference in size.

Assessment: Before proceeding, we would like to remind the reader on the properties of growth, discussed in Law III of self-regulation: the changes are small, come with spike patterns between zero and non-zero deltas and the average value of growth is very close to zero (from the positive side).

Concerning the ripples after large changes, we can detect several patterns. Observe Fig. 9, depicting attribute growth for the MediaWiki dataset. Due to the fact that this involves the growth of attributes, the phenomena are amplified compared to the case of tables. Reading from right to left, we can see that there are indeed cases where a large spike is followed by small or no changes (case 1). However, within the small pool of large changes that exist overall, it is quite frequent to see sequences of large oscillations one after the other, and quite frequently being performed around zero too (case 2). In some occurrences, we see both (case 3).

Concerning the effect of age, we do not see a diminishing trend in the values of growth; however, *age results to a reduction in the density of changes and the frequency of non-zero values in the spikes. This explains the drop of the growth in almost all the studied datasets* (Fig. 5): the linear interpolation drops; however, this is not due to the decrease of the height of the spikes, but due to the decrease of their density.

The heartbeat of the systems tells a similar story: typically, change is quite more frequent in the beginning, despite the

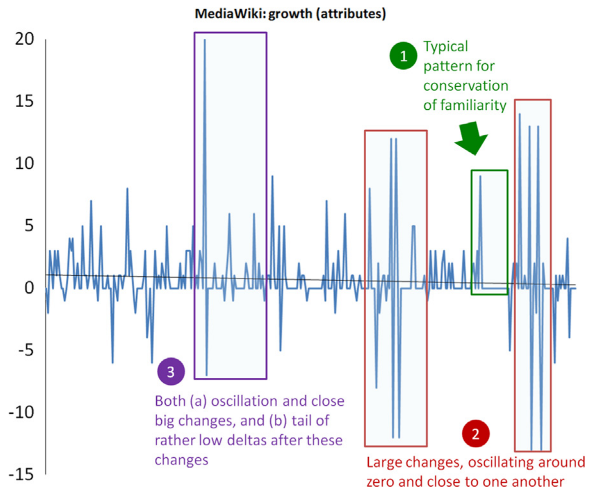


Fig. 9. Different patterns of change in attribute growth of MediaWiki (over version-id, concealed for fig. clarity).

fact that existence of large changes and dense periods of activities can occur in any period of the lifetime. Figs. 2–4 clearly demonstrate this by combining schema size and activity. This trend is typical for almost all of the studied databases. phpBB is the only exception, demonstrating increased activity in its latest versions with the schema size oscillating between 60 and 63 tables, which is actually a very small difference (as all figures are fitted to show the lines as clearly as possible, they can be deceiving as to the amount of change – phpBB is such a case).

Concerning the validity of the law, we *believe that the law is possible but not confirmed*. The law states that the growth is constrained by the need to maintain familiarity. However, the peculiarity of databases, compared to typical software systems, is that there can be other good reasons to constrain growth, such as the high degree of dependence of other modules from the database. Therefore, conservation of familiarity, although important, cannot solely justify the limited growth. The extent of the contribution of each reason is unclear.

4.2.3. *Law of Conservation of Organizational Stability (Law IV)*

The fourth law of software evolution is known as the law of "Conservation of Organizational Stability" also known as law of the "invariant work rate".

> The work rate of an organization evolving an E-type software system tends to be constant over the operational lifetime of that system or phases of that lifetime.

This is the only law with a fundamental change between the two editions of 1996 and 2006. The previous form of the law did not recognize phases in the lifetime of a project (the average effective global activity rate in an evolving E-type system is invariant over product lifetime). Plainly put, the law states that the impact of any managerial actions to improve productivity is balanced by the increasing complexity of software as time passes as well as the role of forces external to the software (availability of resources, personnel, etc.).

Metrics for the assessment of the law's validity: As [23] excellently states, it is very hard to assess effort from the data that we can typically acquire from a project, as "effort does not equate progress". Therefore, we can only approximate the work rate by observing the published versions of a system. Possible metrics [23] include (i) the number of changes per version, (ii) the average number of changes per day, and (iii) change and growth ratios.

To validate the law of conservation of organizational stability, we need to establish that the project's lifetime is divided into phases, each of which (a) demonstrates a constant growth and (b) is connected to the next phase with an abrupt change. Moreover, abrupt changes should occur from time to time and not all the time (resulting in extremely short phases).

Assessment: *If we focus on the essence of the law*, *we can safely say that it does not hold*. The heartbeats of Figs. 2–4 and the arbitrary sequencing of spikes and calmness (Figs. 5, 9) make it impossible to speak about constant growth, even in phases. The open-source nature of our cases plays a role to that too [23].

4.3. *Perfective maintenance for schema evolution*

Lehman has indicated the battle between two antagonizing processes over a fixed amount of resources for the maintenance of software [14]: on the one hand, the need to evolve the system (system growth) and on the other the "anti-regressive" effort to attack the growing complexity of the system. To achieve this, *perfective maintenance* must be performed from time to time, in order to remove redundant code, to restructure code for better maintainability and comprehension, to document the code, etc. As [9] puts it: "these activities have minor or no impact in functionality, performance or other properties of the software in execution". In this subsection, we are interested in the perfective maintenance part and we adopt the [32] definition (emphasis is ours): "*modification of a software product after delivery to provide* enhancements for users, *improvement of program documentation*, and *recoding to improve software* performance, *maintainability or other software attributes*".

4.3.1. *Law of Increasing Complexity (Law II)*

The second law of software evolution is known as the law of "Increasing Complexity".

> As an E-type system is changed its complexity increases and becomes more difficult to evolve unless work is done to maintain or reduce the complexity.

The law states that complexity increases with age, unless effort is taken to prevent this. The rationale behind verifying the law dictates the observation of (a) an increasing trend in complexity of a software system, battled by (b) a perfective maintenance activity that attempts to reduce it and demonstrated by drops in the system size and rate of expansion.

Metrics for the assessment of the law's validity: Since we will ultimately resort to measurements for verifying the law, before proceeding further, we need to confront a fundamental problem: *the law's definition – as it stands – requires a more precise definition of complexity*. Unfortunately, complexity is a meta-property, practically involving a wide spectrum of specific measurable properties of software. To give an example, Fenton and Pfleegler [33] mention four kinds of complexity: (i) *problem complexity* (computational complexity of the underlying problem), (ii) *algorithmic complexity* (of the algorithm eventually implemented to solve the problem), (iii) *structural complexity* (typically measured as the control flow or class hierarchy or modularity structure) and (iv) *cognitive complexity* (measuring the effort required to understand the software). Lehman and Ramil [9] take a more process-oriented approach and refer to *application and functional complexity*, *specification and requirements complexity*, *architectural complexity*, *design and implementation complexity* and *structural complexity*.

Unfortunately, all the above are very hard to define and measure, especially if measurement is to be performed on evidence automatically extracted from electronic logs or version management systems. The automatic isolation of the subset of changes that pertain to perfective maintenance is an interesting and vast topic of research; for the moment, however, it appears that we will have to resort to approximations. Related literature is based on such approximations (see for example, [34]). Notably, in the latest of Lehman's series of papers, the law is supported via rationalization: the complexity increase that age brings to a system is considered responsible for the decline of the growth ratio over time (laws V and VI).

To surpass all these difficulties, we will try to assess the validity of the law based on the combination of the following observations:

First, we will focus on the essence of the law: ultimately, the law requires identifying releases or versions where perfective maintenance is performed. To actually achieve with 100% certainty would require some project management documentation that this is performed. Thus, we resort to the closest possible approximation and try to *detect versions with drops in the size and the growth of the system*. Assuming that the overall trend of the system is to grow, the existence of such points from time to time will give a strong indication of the law.

A second indication for the validity of the law is *the respect of the VIII law of feedback*, i.e., the existence of a regressive formula to which the size of the system conforms. The validity of this law would strongly insinuate the existence of a feedback-based system and therefore, the existence of negative feedback as discussed in this second law of evolution.

Third, we take a definition already found in Lehman [7,34] and *attempt to approximate the measurement of complexity as the fraction of the evolution-affected relations (i.e., the number of relations modified or added to the schema) between two subsequent versions of the schema over the difference in the number of relations of the involved versions*. This formula approximates how much effort has been invested in expanding the system over the actual difference achieved (large values demonstrate too much effort for too small change). So, for each transition, we approximate the complexity of the original schema by dividing the extent of the involved changes over the actual increment of the schema size. To understand this better, assume that we compare two transitions with the same denominator (i.e., difference in the number of relations); if one transition had more relations updated than the other, it means we paid more effort for this transition, and thus, we assume that the starting complexity is higher. More precisely, we divide the effort (number of relations that we modified in any way in a revision), by the growth (size of the result in that revision). In case the denominator is zero, we have no escape than to define complexity as zero (which is another approximation we cannot avoid).

$$complexity_i \sim \frac{relations\ handled}{|S_i - S_{i-1}|} \tag{6}$$

Assessment: Related literature typically speaks for increasing complexity [7–9,23], although there have been counter-arguments for the case of open source software [35]. In our case, *in all the datasets but phpBB, complexity, as defined in the previous paragraph, does not increase*[13] (see Fig. 10, where a linear interpolation of complexity is also depicted). The phenomenon must be coupled with the drop in change density (Law V) and although we cannot provide undisputable explanation, we offer the synergy of two causes: (a) the increasing dependence of the surrounding code to the database that makes developers more cautious to perform schema changes as they incur higher maintenance costs and (b) the success of the perfective maintenance, which results in a clean schema, requiring less corrective maintenance in the future.

Although we cannot confirm or disprove the law based on undisputed objective measurements, we have indications that the second law partially holds, albeit with completely different connotations than the ones reported by Lehman for typical software systems: in the case of database schemata, complexity, when measured as the fraction of expansion effort over actual growth, drops.

4.3.2. Law of Declining Quality (Law VII)

The seventh law of software evolution is known as the law of “Declining Quality”.

> Unless rigorously adapted and evolved to take into account changes in the operational environment, the quality of an E-type system will appear to be declining.

The main idea behind this law concerns the fact that the software will each time be based on assumptions on the user requirements or the real world environment that will progressively be invalid. As assumptions are invalidated, action must be undertaken to maintain the affected software parts in order to reflect the actual user needs. Thus, the aging of the system, along with the increase in complexity, also calls for a reestablishment of assumptions and functionalities to serve the users' needs. Ref. [14] specifically refers to the external quality of a software system, practically expressing a system's quality as ‘user satisfaction’. However, this point of view is drastically different in [9], where the viewpoint on quality is generalized to all possible kinds of quality an organization might deem necessary (based on the viewpoint of users,

[13] In our CAiSE'14 paper [31], we erroneously refer to BioSQL instead of phpBB.

managers, developers, each carrying his own interpretation and measures).

Metrics for the assessment of the law's validity: Possible metrics [23] for the internal quality of typical software systems include (i) the number of known defects associated with each version, (ii) defect density for each version, (iii) percentage of modules whose bodies have been changed. Much like the authors of [23], however, we are not really in a position to fully automate the accurate measurement of external quality as perceived by the end users, the management, etc. It is noteworthy that Lehman and Fernandez-Ramil [15,9] avoid giving any other support to the law than a logical proof: as the system expands over time, its complexity rises and thus the addressing of user requirements and removal of defects becomes more and more difficult, unless work is done to confront the phenomenon (the decline in software quality with age appears to relate to a growth in complexity that must be associated with aging).

Assessment: We follow [9] and *use logical induction to assess whether the law holds; specifically, we can assume that the law holds if it is strongly established that the laws of feedback* (*III*, *VIII*) *and complexity* (*II*) *hold*.

We have already demonstrated that the rationale behind complexity increase is not supported by our observations. At the same time, we cannot assess schema quality with undisputed means. Therefore, *we cannot confirm or disprove the law based on undisputed objective measurements*.

4.4. Threats to validity

In this subsection, we discuss threats to the validity of our conclusions. We structure our deliberations around three kinds of validity threats, specifically, construct validity, assessing the appropriateness of our measures, internal validity, assessing the possibility that cause–effect relationships are produced on an erroneous interpretation of causality, and external validity, assessing the extent to which our results can be generalized.

4.4.1. Construct validity

Construct validity concerns the appropriateness of the employed measures for the theoretical constructs they purportedly assess. In our case, to assess construct validity, we review the appropriateness of the metrics used for each law, also with a view to the metrics used in the studies of software evolution. Fig. 11 summarizes our assessment.

I. *Continuing change*: As the goal is to establish the continuity of change, the usage of the (accurately measured) heartbeat raises no concern about its appropriateness and the validity of our results.

II. *Increasing complexity*: The main metric to assess this law is the schema complexity. As we mentioned before, we do not have a way to accurately measure the complexity of a database schema as similar studies have done with software's complexity. We approximate the complexity with the effort spent between two schema versions divided by the increment in size between those versions. The later can be accurately measured but this is not the case with the effort. Effort cannot be measured from the data that we have extracted for the databases that we studied. The only accurate way to measure effort would be to have the actual man-hours that every developer has spent in the development of the database. Moreover, given the fact that databases are parts of larger software ecosystems, the possibility of accurately assessing

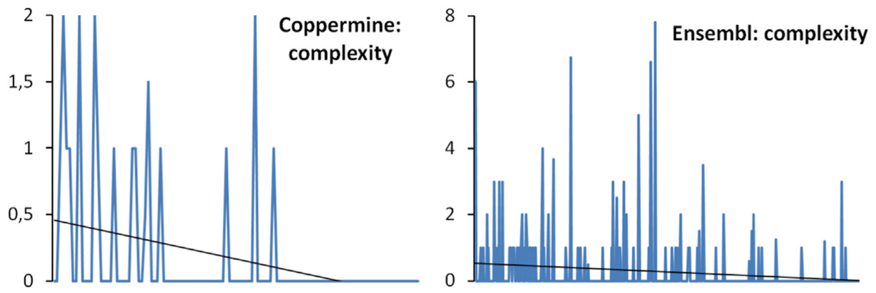
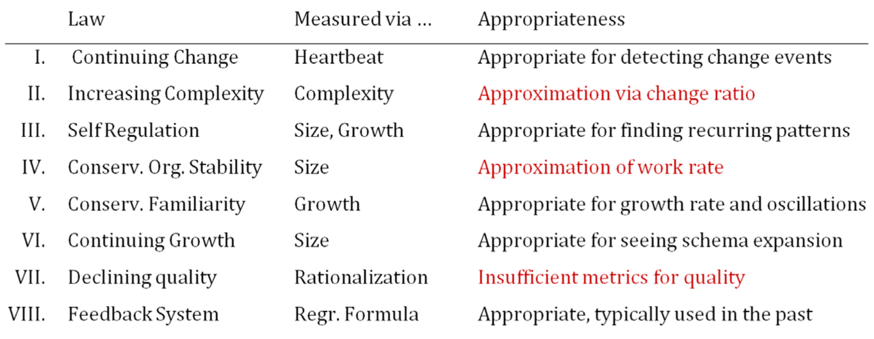


Fig. 10. Complexity for Coppermine and Ensembl (over version-id, concealed for clarity).



	Law	Measured via ...	Appropriateness
I.	Continuing Change	Heartbeat	Appropriate for detecting change events
II.	Increasing Complexity	Complexity	Approximation via change ratio
III.	Self Regulation	Size, Growth	Appropriate for finding recurring patterns
IV.	Conserv. Org. Stability	Size	Approximation of work rate
V.	Conserv. Familiarity	Growth	Appropriate for growth rate and oscillations
VI.	Continuing Growth	Size	Appropriate for seeing schema expansion
VII.	Declining quality	Rationalization	Insufficient metrics for quality
VIII.	Feedback System	Regr. Formula	Appropriate, typically used in the past

Fig. 11. Summary of measures employed per law and their appropriateness.

effort would require a measure able to differentiate the work done on the database and the work pertaining to the rest of the software system – a possibility which we dim quite slim, in fact. On the other hand, the reasoning behind the formula used makes sense and it is consistent with the related literature. Overall, the complexity, as we approximate it, poses a threat to our construct validity that we cannot ignore; to a large extent, this is also due to the abstract wording of the law. This is also the reason why we are very skeptic towards verifying the validity of the law in the case of schema evolution. Future work needs to be invested in the area for a more solid grounding of automated complexity assessment.

III. *Self regulation*: To assess this law, we used schema size and growth as measures. Both metrics can be accurately measured. The usage of the measure is consistent with the bibliography and the intuition behind the law.

IV. *Conservation of organizational stability*: The involved metric in order to assess this law is the work rate (and the existence of periods during which it remains constant). As previously mentioned, work rate cannot be easily measured, based on the available information. To this end, we primarily use schema growth as an approximation of output, and secondarily the heartbeat as an approximation of activity, both of which are accurate. Overall, we are satisfied with our choice, as it appears that this is the best possible approximation we can get from automatically extracted data; at the same time, we have to acknowledge that it is an approximation and not an undisputed measurement of the work rate.

V. *Conservation of familiarity*: The metric used for the assessment of this law is growth, which is accurately measured. On the other hand, we have no way to indisputably know the exact mechanics behind the observations; hence, despite the accuracy of the observations, the law requires further elaboration.

VI. *Continuing growth*: For this law, we employed schema size again, which is accurately extracted by our tools and fit for assessing the law.

VII. *Declining quality*: As schema quality is not clearly defined in the area of databases, the assessment of quality via metrics requires specific studies on the topic, before we are able to converge to a widely accepted solution. Rationalization about the law has typically been used in the related literature as a solution to the problem.

VIII. *Feedback system*: The main measure we used for assessing this law is the estimated size of the database schema. This measure has previously been used in the case of software evolution, again with an approximation for the measurement of effort. However, the regression formula used is consistent with its usage in the bibliography (albeit with novelty in terms of the memory of the feedback) and all the results in all datasets are surprisingly consistent. Therefore, we believe that the specific formulae used pose no threat to validity, although a better understanding of the mechanics behind the feedback mechanism have to be part of future studies.

4.4.2. Internal validity

Internal validity refers to the case where a conclusion on the behavior of a dependent variable is made as a cause–effect relationship with an independent variable. We are very careful to treat our observations only as such and avoid relating the observed phenomena with specific causes without supporting evidence.

Having said that, we extend the discussion, as the observant reader might be tempted to introduce a cause–effect relationship between age (as a cause) and the following phenomena: (a) dropping density of change, (b) dropping complexity, and (c) size growth in the long run. We conjecture (but cannot prove) that we could attribute the behavior of density and complexity to the existence of a confounding variable: schema quality, improving over time due to perfective maintenance and causing the observed behavior. Still, this remains to be proved with undisputed data and metrics. For size, the confounding variable is user requirements for more information capacity; although reasonable enough (in our minds, practically certain), this is also a topic to be proved indisputably by dedicated studies.

4.4.3. External validity

External validity refers to the possibility of generalizing the findings of a study to a broader context. Concerning the *external validity* of our study, we repeat that its context concerns *the study of the evolution of the* <u>*logical*</u> *schema of databases in* <u>*open-source*</u> *software*. We avoid generalizing our findings to databases operating in closed environments and we stress that our study has focused only on the logical structure of databases, avoiding physical properties (let alone instance-level observations).

Concerning the validity of our study within this context, we believe we have provided a safe, representative experiment. In this study, we have targeted a significant number of database schemas that serve different purposes in the real world and come with a quite broad range of time spans. Concerning the time span, the schemas collected had an adequate number of versions from rather few (40) to quite many (500+). Despite these degrees of variability, our findings are consistent in practically all of the datasets (with few exceptions that we mentioned). Thus we believe that the case of logical database schema in open source software is well represented.

On the other hand, we would be hesitant to generalize our findings in databases in closed software or outside the scope of the logical schema. Open-source software comes with a larger development community, and less control on the development effort. This is not the case for closed software, especially when dealing with mission critical components like databases. At the same time, we have not worked with the information concerning the physical schema or the extension of the studied databases and thus, we would take the opportunity to warn the reader not to generalize the results outside the scope of a schema's information capacity as expressed by the logical-level schema.

5. Discussion

In this section, we summarize fundamental *observations* and *patterns* that have been detected in our study. We intentionally avoid the term *law*, as we do not have unshakeable evidence for their explanation: apart from the *empirical grounding*, due to a very large amount of datasets that obey the same patterns (which we believe we have fairly attained), we would require an undisputed *rationalized grounding*, i.e., a clear explanation of the underlying mechanism that guides them, also established on measured, undisputed facts.

In case the reader has skipped our discussion of threats to validity, we clarify once more that the context under which our observations are made concerns *the study of the evolution of the logical schema of databases in open-source software*. In all our subsequent deliberations, we take the above context as granted and avoid repeating it for reasons of better presentation of our results.

Before proceeding, however, to our conclusions, we devote the first part of this subsection to a discussion on the validity of the problem per se.

5.1. Does the problem make sense in the first place?

We start with a *fundamental inquiry*: Is it meaningful to try assessing Lehman's laws for schema evolution in the first place? Does it make sense to try to observe evolutionary patterns in the way schemata evolve by following Lehman's method and laws?

Surely, there are fundamental differences between the general case of E-type software systems and databases in open-source systems. First, whereas software systems export *functionality* to their users, databases, on the contrary, export *information capacity*, i.e., the ability to store data and answer queries. Second, databases are not complete and independent software systems but parts of larger information systems. Is it then meaningful to pursue this research?

Again, let us revisit the fundamental lesson learned by Lehman's laws: software systems are complex, multi-level systems, involving several stakeholders, that have to evolve or face eviction; this evolution is governed by the antagonism between (a) positive feedback, pushing the system to adapt to new environments and add new functionalities according to the users' needs and (b) negative feedback, that constrains the uncontrolled growth and complexity of the system, by imposing perfective maintenance actions that result in an improved, more maintainable internal structure of the system.

Can we replace the term ‘software systems’ with ‘database’ in the above wording? We believe we can, and the fundamental reason is that the antagonism between positive and negative feedback is there too. On the one hand, a database schema has to obey the part of the positive feedback and its moderators need to adapt, tune and expand it over time (and this concerns all kinds of databases, as well as the ones involved in open-source software). This concerns both the expansion due to user requirements concerning the availability of information and the adaptation to new environments. At the same time, growth cannot be unconstrained: developers of open-source software are highly sensitive and attentive when it comes to database-related code, as changes in the database can incur both syntactic and semantic failures. Thus, it would be reasonable to expect that leaving the schema grow without any complexity control, especially in an open-source environment where developers are not organized in a strict hierarchy, can result to maintenance nightmares. The a posteriori observations verify this intuition: we do observe schema size contractions, where renamings, restructurings and removal of tables and attributes are evident in an attempt to keep schemata clean, understandable and well-structured.

Are databases, then, mini E-type systems with a life of their own? We should be clear that we do not postulate that databases can be completely isolated from the rest of their surrounding ecosystem. Still, studying schema evolution in an attempt to discover regularities and patterns is certainly worth the effort, given the high degree of dependence of the rest of the code over the database structure. With the benefit of the hindsight, we do believe that considering the laws of Lehman as a starting point for the study of schema evolution has been a legitimate and rewarding effort as it revealed both commonalities (mainly due to the same fundamental feedback mechanism) and differences (due to the specificities of the database case) with the general theory of Lehman's laws.

5.2. Major findings

In this section, we provide a critical discussion of our findings, accompanied by concise summaries, where we also annotate each of our observations with reference to the law where we have discussed it in detail. Fig. 12 further distils these findings in a single table.

5.2.1. Is the process of schema evolution behaving like a feed-back based system? (hypothesis of the feedback-based process)

We believe that we can indeed claim that schema evolution is guided by a feedback based mechanism. Positive feedback brings the need to increase the information capacity of the database, resulting in expansion of the number of relations and attributes over time. At the same time, there is negative feedback too, from the need to do some house-cleaning of the schema for redundant attributes or restructuring to enhance schema quality. We have also observed that the inverse square models for the prediction of size expansion holds for all the eight schemata that we have studied. However, we do not come with a good explanation as to why this holds. The supporting observations in this context can be listed as follows:

- As an overall trend, the information capacity of the database schema is enhanced – i.e., the size grows in the long term (VI).
- The existence of perfective maintenance is evident in almost all datasets with the existence of relation and attributes' removals, as well as observable drops in growth and size of the schema (sometimes large ones). In fact, growth frequently oscillates between positive and negative values (III).
- The schema size of a certain version of the database can be accurately estimated via a regressive formula that exploits the amount of changes in recent, previous versions (VIII).

As in all feedback-based systems, the negative feedback prevents the uncontrolled growth and retains the quality of the schema at a high level, allowing thus the subsequent

LAW & RESEARCH QUESTIONS	MEASURES	FINDINGS & COMMENTS	DATASETS
I Continuing change The schema is continually adapted	Heartbeat	The schema is adapted in the long run, albeit not continually, but in focused periods of modification	All datasets abide by the law, with two exceptions: - BioSQL, with a "sleep" of some years - phpBB with a turbulence period
III Self-regulation - Schema size expands with recurring patterns of smooth expansion, interrupted by abrupt change - Existence of shrinking versions (negative feedback)	Size	- Patterns of change include (a) expansion, (b) shrinking and (c) stability; differently from the expression of the law - Perfective maintenance is evident	All datasets without exceptions
- Change normally distributed around an average value	Growth	- Growth is small, typically close to zero, following a Zipfian model - Oscillations of large size do exist	
VIII Feedback System We can estimate the schema size via regressive formula	Regression analysis	Size estimation can be achieved; out of the different alternatives for effort estimation, the ones with small time window work better	All datasets without exceptions
VI Continuing growth The schema size is increasing in the long run	Size	Size increases in the long run, indeed, albeit not continually, but in focused periods of modification	All datasets without exceptions
V Conservation of familiarity - The average growth between versions is slowly declining	Heartbeat, Size	- The linear interpolation of growth typically drops or stays stable; importantly, the frequency of change declines	All datasets except for Atlas and BioSQL (with some extra activity in the end of their lifetimes)
- What happens after excessive changes?	Growth	- Spikes are followed by all possible combinations (calmness, other spikes, large oscillations around zero)	All datasets exhibit various patterns
IV Invariant work rate Avg. work-rate is constant within phases of smooth growth, connected with bursts of effort	(approximate) Heartbeat & Size	There is are no phases of constant growth; albeit periods of stability connected via focused periods of modifications	All datasets without exceptions
II Increasing complexity Complexity increases over time	(approximate) Schema Complexity	Complexity drops	All datasets except for phpBB (having a turbulence period in the end)
VII Declining Quality Quality declines over time	Conjecture by logical induction	Impossible to hold as valid, as complexity (albeit approximated) seems to drop	-

Fig. 12. Summary of research questions, findings and validity over the datasets.

releases to operate smoothly. This is a true sign of *stability* : the system is maintained adequately to minimize the effects of its unavoidable subsequent modifications and continue evolving smoothly.

Overall, we can state: *Schema evolution demonstrates the behavior of a stable, feedback-regulated system, as the need for expanding its information capacity to address user needs is controlled via perfective maintenance that retains quality; this antagonism restrains unordered expansion and brings stability.*

5.2.2. Hypothesis of schema size expansion (and properties of its growth)

The size of the schema expands over time, albeit with versions of perfective maintenance due to the negative feedback. As already mentioned, the inverse square model seems to work.

The growth of the database schema does not follow a pattern of smooth growth – even considering the amendment where phases of constant growth are assumed. The expansion is mainly characterized by three kinds of phases, including (i) abrupt change (positive and negative), (ii) smooth growth, and (iii) calmness (meaning large periods of no change, or very small changes). We observe that in the case of schema evolution, *the schema's growth* (i.e., its change from one version to the following) *mainly occurs with spikes oscillating between zero and non-zero values. The changes are typically small*, following a Zipfian distribution of occurrences, with high frequencies in deltas that involved small values of change, close to zero.

At the same time, in contrast to the case of software systems, *we observe a very strong inclination to avoid changes to the database schema*. Change in the database impacts surrounding code, so the change is constrained by the need to minimize this impact. So, we frequently see versions with no change to the information capacity of the schema and large time periods where the schema is still (or almost still). Bear in mind that we monitor only the subset of versions that pertain to the database schema and ignored any versions where the information system surrounding the database changed while the schema remained the same. This enforces our argument for the tendency towards stillness.

Although we do not believe conservation of familiarity to be the only cause, we see that the feedback mechanism of the evolution demonstrates *a reduction in the density of changes as the schema ages*. We also observe unexpected patterns of changes with sequences of high spikes, sometimes oscillating around zero. Such patterns require further investigation for their verification and explanation. The average growth is close to zero, and with the tendency to drop as time passes, not due to the diminishing of the (already small) deltas, whenever they occur, but mainly due to the diminishing of their density.

Concerning the *size* of the system, our supporting evidence has been already summarized via laws VI and VIII (see the previous paragraph). Concerning the *heartbeat* of the system, our supporting evidence for the above statements can be listed as follows:

- The database is not continuously adapted, but rather, alterations occur from time to time (I).
- Change does not follow patterns of constant behavior (IV).
- Age results in a reduction of the density of changes to the database schema in most cases (V).

Concerning the *growth* of the system, our supporting evidence for the above statements can be listed as follows:

- Growth is typically small in the evolution of database schemata, compared to traditional software systems (III). The distribution of occurrences of the amount of schema change follows a Zipfian distribution, with a predominant amount of zero growth in all datasets. Plainly put, there is a very large amount of versions with zero growth, both in the case of attributes and in the case of tables. The rest of the frequently occurring values are close to zero, too.
- The average value of growth is typically close to zero (although positive) (III) and drops with time, mainly due to the drop in change density (V).

5.2.3. *Hypothesis of perfective maintenance to fight complexity and user dissatisfaction*

We also believe that *there is sufficient evidence to support the claim that perfective maintenance is part of the process. This is mainly demonstrated by the drops in the schema size as well as the drops in activity rate and growth with age. In fact, growth frequently oscillates between positive and negative values* (*III*). Thus, based on simple reasoning, one can accept the wording of Lehman's laws on negative feedback, as they both state that quality (internal and external) declines unless confronted.

However, despite the adoption of the hypothesis for a feedback-based mechanism, we cannot adopt the corroborating observations of the related literature for software systems that accompany the two laws of negative feedback (II and VII). In the systems we have studied we observe that *age results in a reduction of the complexity to the database schema* (*II*), although we need to remember that the measurement of complexity is an approximation. The interpretation of the observation is that perfective maintenance seems to do a really good job and complexity drops with age (in sharp contrast to what is observed in the related literature for software systems where more and more effort is devoted to battle complexity). Also, in the case of schema evolution, activity is typically less frequent with age. Although one can attribute this to the inefficacy of the approximating measure, we anticipate that it should mainly be attributed to the truth lying in the essence of law II: "complexity increases unless work is done to reduce it". We conjecture that due to the criticality of the database layer in the overall information system, this process is done with care and achieves the reduction of complexity over time, coming hand in hand with the strong tendency towards minimum or no changes to the schema.

As for law VII, as already mentioned, we are even more hesitant to adopt it, as we are already in doubt towards internal quality and have no actual evidence as to what happens with external quality.

Overall: *although our research seems to keep the negative feedback laws in place in the case of schema evolution*, *this is done with* (a) *a degree of uncertainty and* (b) *with the strong indication of fundamental differences with E-type program evolution*. We would not be surprised if future research establishes with more certainty that the feedback mechanism for schema evolution improves the quality and complexity of a database as time passes.

5.3. *Opportunities for future work*

There are several opportunities for follow-up work. As one would normally expect, verifying the findings of this study with more datasets can further solidify our confidence to them. The extension of this work to evolution histories of proprietary databases in closed environments, over large periods of time, would be of extreme value; albeit one can only be pessimistic on the possibility of obtaining such data and being able to publish them. Novel developments in database technology allow the extension of this kind of study to non-relational data too. This includes all kinds of semi-structured data (evolution of XML data alone is a vast area of research, where the nesting of the elements provides transformations of the schema that are not present in the relational case), but also, the so-called "NoSQL" data, where structures like graphs and text evolve over time. In the latter case, the identification of patterns in the evolution of the data at the instance level is clearly a challenging topic of research.

A second large area of research concerns the identification of patterns in the correlation of the evolution of the database and the evolution of the surrounding applications. This involves both the alignment of the application code to the new schema and, as a reviewer of this paper has pointed out, possible workarounds in the code to avoid modifying the database. Even more challenging is the relationship of user requirements to database evolution. Remember that in order to be able to come up with results in long histories with many versions, automated processing of the available data is paramount. The possibility of automating the processing of tickets, bug reports and to-do lists in a way that can be correlated to the subsequent evolution of the database is a topic with a significant amount of technical challenge.

At the same time, the techniques used in this study provide opportunities for improvement. A first area of future research concerns the findings of aging and complexity (Law II). We need to establish better measures for complexity of database schemata and see how this complexity behaves over time. Similar considerations hold for estimating effort and work-rate by exploiting the available information in the software repositories as automatically as possible.

Finally, one should also recognize that the search for more patterns than the ones offered by Lehman's laws, via traditional or novel pattern detection mechanisms, is another important possibility for future work. Already, the observation of patterns of growth (Laws III and V), or patterns in the heartbeat of the evolution, are open issues worth investigating. Going further than that, identifying which tables are more liable to change in the future and how, or how the effort around schema evolution can be planned in advance by studying the available data are research questions with great value both for developers, who can tailor the code to be as loosely coupled as possible to the most unstable parts of the database, and project managers, who can estimate where change will be directed. We hope that in the context of such endeavors, the publicly available datasets of this paper (https://github.com/DAINTINESS-Group) can serve the research community.

Acknowledgment

We would like to thank the anonymous reviewers of both [31] and this paper for their useful comments.

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

References

[1] D. Sjøberg, Quantifying schema evolution, Inf. Softw. Technol. 35 (1) (1993) 35–44.
[2] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou, Metrics for the prediction of evolution impact in ETL ecosystems: a case study, J. Data Semant. 1 (2) (2012) 75–97.
[3] C. Curino, H.J. Moon, L. Tanca, C. Zaniolo, Schema evolution in wikipedia: toward a web information system benchmark, in: Proceedings of 10th International Conference on Enterprise Information Systems (ICEIS), 2008.
[4] D.-Y. Lin, I. Neamtiu, Collateral evolution of applications and databases, in: Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshops (IWPSE), 2009, pp. 31–40.
[5] S. Wu, I. Neamtiu, Schema evolution analysis for embedded databases, in: Proceedings of the 27th IEEE International Conference on Data Engineering Workshops (ICDEW), 2011, pp. 151–156.
[6] D. Qiu, B. Li, Z. Su, An Empirical analysis of the co-evolution of schema and code in database applications, in: Proceedings of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), 2013, pp. 125–135.
[7] L.A. Belady, M.M. Lehman, A model of large program development, IBM Syst. J. 15 (3) (1976) 225–252.
[8] M.M. Lehman, J.C. Fernandez-Ramil, P. Wernick, D.E. Perry, W.M. Turski, Metrics and laws of software evolution—the nineties view, in: Proceedings of the 4th IEEE International Software Metrics Symposium (METRICS), 1997, pp. 20–34.
[9] M.M. Lehman, J.C. Fernandez-Ramil, Rules and Tools for Software Evolution Planning and Management, Software Evolution and Feedback: Theory and Practice, Wiley, Chichester, West Sussex, England, 2006.
[10] 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 (2) (2013) 1–28.
[11] M. Wermelinger, Y. Yu, A. Lozano, Design principles in architectural evolution: a case study, in: Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM), 2008, pp. 396–405.
[12] Z. Xing, E. Stroulia, Analyzing the evolutionary history of the logical design of object-oriented software, IEEE Trans. Softw. Eng. 31 (10) (2005) 850–868.
[13] M. Lehman, Programs life cycles and laws of software evolution, Proc. IEEE 68 (9) (1980) 1060–1076.
[14] M.M. Lehman, Laws of software evolution revisited, in: Proceedings of 5th European Workshop on Software Process Technology (EWSPT), 1996, pp. 108–124.
[15] M.M. Lehman, J.C. Fernandez-Ramil, D.E. Perry, On evidence supporting the FEAST hypothesis and the laws of software evolution, in: Proceedings of the 5th IEEE International Software Metrics Symposium (METRICS), 1998, pp. 84–88.
[16] S.S. Pirzada, A statistical examination of the evolution of the unix system (Ph.D. thesis), Imperial College, University of London, 1988.
[17] N.T. Siebel, S. Cook, M. Satpathy, D. Rodríguez, Latitudinal and Longitudinal Process Diversity, J. Softw. Maint. Res. Pract. 15 (1) (2003) 9–25.
[18] M.J. Lawrence, An examination of evolution dynamics, in: Proceedings of the 6th International Conference on Software Engineering (ICSE), 1982, pp. 188–196.
[19] M.W. Godfrey, Q. Tu, Evolution in open source software: a case study, in: Proceedings of the 16th IEEE International Conference on Software Maintenance (ICSM), 2000, pp. 131–142.
[20] 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), 2001, pp. 103–106.
[21] G. Robles, J.J. Amor, J.M. Gonzalez-Barahona, I. Herraiz, Evolution and growth in large Libre software projects, in: Proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE), 2005, pp. 165–174.
[22] S. Koch, Software evolution in open source projects: a large-scale investigation, J. Softw. Maint. Evol. 19 (6) (2007) 361–382.
[23] G. Xie, J. Chen, I. Neamtiu, Towards a better understanding of software evolution: an empirical study on open source software, in: Proceedings of the 25th IEEE International Conference on Software Maintenance (ICSM), 2009, pp. 51–60.
[24] 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 10th European Conference on Software Maintenance and Reengineering (CSMR), 2006, pp. 206–213.
[25] R. Vasa, Growth and change dynamics in open source software systems (Ph.D. thesis), Swinburn University of Technology, Australia, 2010.
[26] A. Israeli, D.G. Feitelson, The Linux kernel as a case study in software evolution, J. Syst. Softw. 83 (3) (2010) 485–501.
[27] C.A. Curino, H.J. Moon, C. Zaniolo, Graceful database schema evolution: the PRISM workbench, Proc. VLDB Endow. 1 (2008) 761–772.
[28] C. Curino, H.J. Moon, A. Deutsch, C. Zaniolo, Automating the database schema evolution process, VLDB J. 22 (1) (2013) 73–98.
[29] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou, Policy-regulated management of ETL evolution, J. Data Semant. 13 (2009) 147–177.
[30] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou, HECATAEUS: regulating schema evolution, in: Proceedings of the 26th IEEE International Conference on Data Engineering (ICDE), 2010, pp. 1181–1184.
[31] I. Skoulis, P. Vassiliadis, A. Zarras, Open-source databases: within, outside, or beyond Lehman's laws of software evolution? in: Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE), 2014, pp. 379–393.
[32] IEEE, Guide to the Software Engineering Body of Knowledge (v. 3.0), IEEE Computer Society, 2014, available at 〈http://www.computer.org/portal/web/swebok〉, retrieved at 08 July 2014.
[33] N.E. Fenton, S.L. Pfleeger, Software Metrics—A Practical and Rigorous Approach, International Thomson, Boston, MA, USA, 1996.
[34] M.M. Lehman, J.F. Ramil, Software Evolution, in: STRL Annual Distinguished Lecture, De Montfort University, Leicester, 20 December 2001, available at 〈http://www.eis.mdx.ac.uk/staffpages/mml/feast2/papers.html〉, 〈http://www.eis.mdx.ac.uk/staffpages/mml/feast2/papers/pdf/690c.pdf〉, 〈http://www.eis.mdx.ac.uk/staffpages/mml/feast2/papers/pdf/jfr103c.pdf〉.
[35] J. Fernández-Ramil, A. Lozano, M. Wermelinger, A. Capiluppi, Empirical studies of open source evolution, in: Software Evolution, 2008, pp. 263–288.