Analysis of Schema Evolution for Databases in Open-Source Software

Ioannis Skoulis

Master Thesis

Ioannina, September 2013

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

ΑΝΑΛΥΣΗ ΤΗΣ ΕΞΕΛΙΞΗΣ ΣΧΗΜΑΤΟΣ ΓΙΑ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ ΣΕ ΛΟΓΙΣΜΙΚΟ ΑΝΟΙΧΤΟΥ ΚΩΔΙΚΑ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τον

Ιωάννη Σκουλή

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Σεπτέμβριος 2013

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Ioannis Skoulis
MSc, Department of Computer Science and Engineering
University of Ioannina, Greece
September 2013
Analysis of Schema Evolution for Databases in Open-Source Software.
Supervisor: Panos Vassiliadis

Like all software systems, databases are subject to evolution as time passes. The impact of this evolution is tremendous as every change to the schema of a database affects the syntactic correctness and the semantic validity of all the surrounding applications and de facto necessitates their maintenance in order to remove errors from their source code. In this MSc Thesis, we perform a large-scale study for the schema evolution of databases that are part of larger open source projects, publicly available through open source repositories. The goal of our study is to assess the applicability of Lehman's laws of software evolution to databases in open-source software. This set of eight laws of evolution is a well-established set of observations (matured during the last forty years) on how the typical software systems evolve. However, the applicability of these laws on databases has not been studied so far. To this end, we have performed a thorough study on the evolution of several properties of a database (including size, growth, and amount of change per version, in terms of both relations and attributes) and report on the validity of each law on the grounds of these observations.

ΠΕΡΙΛΗΨΗ ΣΤΑ ΕΛΛΗΝΙΚΑ

Ιωάννης Σκουλης του Αντωνίου και της Ειρήνης
ΜΔΕ, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
Σεπτέμβριος 2013
Ανάλυση της εξέλιξης σχήματος για βάσεις δεδομένων σε λογισμικό ανοιχτού κώδικα.
Επιβλέποντας: Παναγιώτης Βασιλειάδης

Όπως σε όλα τα συστήματα λογισμικού, έτσι και οι βάσεις δεδομένων εξελίσσονται στο πέρασμα του χρόνου. Ο αντίκτυπος αυτής της εξέλιξης είναι εξαιρετικής σημασίας καθώς κάθε αλλαγή στο σχήμα μίας βάσης δεδομένων επηρεάζει την συντακτική ορθότητα καθώς και τη σημασιολογική εγκυρότητα όλων τον εφαρμογών που την περιβάλουν και απαιτεί εκ των πραγμάτων τη συντήρησή τους, προκειμένου να αρθούν τα λάθη από τον πηγαίο τους κώδικα. Σε αυτήν την μεταπτυχιακή εργασία εξειδίκευσης, διεξάγουμε μια μεγάλης κλίμακας μελέτη όσον αφορά την εξέλιξη σχήματος βάσεων δεδομένων που αποτελούν μέρος μεγαλύτερων προγραμμάτων ανοικτού κώδικα, διαθέσιμων στο κοινό μέσω ανοικτών αποθετηρίων. Ο στόχος της μελέτης μας είναι να εκτιμήσει τη δυνατότητα εφαρμογής των νόμων του Lehman για την εξέλιξη λογισμικού, σε βάσεις δεδομένων λογισμικού ανοιχτού κώδικα. Αυτό το σύνολο των οκτώ νόμων της εξέλιξης είναι ένα καλά εδραιωμένο σύνολο παρατηρήσεων (που έχει ωριμάσει κατά τη διάρκεια των τελευταίων σαράντα ετών) σχετικά με το πώς τα συστήματα λογισμικού εξελίσσονται. Ωστόσο, η ισχύς των νόμων αυτών όσον αφορά τις βάσεις δεδομένων δεν έχει μελετηθεί μέχρι σήμερα. Για το σκοπό αυτό, έχουμε πραγματοποιήσει μια εμπεριστατωμένη μελέτη σχετικά με την εξέλιξη των διαφόρων ιδιοτήτων της βάσης δεδομένων (όπως μέγεθος, ανάπτυξη, και το ύψος των αλλαγών ανά έκδοση, τόσο όσον αφορά τις σχέσεις όσο και τα γνωρίσματα αυτών) και αναφέρουμε τα αποτελέσματα σχετικά με την εγκυρότητα του κάθε νόμου με βάση τις εν λόγω παρατηρήσεις.

CHAPTER 1. INTRODUCTION

An elemental characteristic of all software projects is their need to evolve. Such behavior occurs due to several reasons. Firstly, people do not know in advance or are unable to express the exact functionality of a large-scale system, which can only be systematically specified only after significant use of the system at hand. Moreover, the application world is continually changing with new programming styles and techniques added to the developer's portfolio. Therefore, continuous modifications are necessary to ensure that the system complies with its requirements. *Software evolution*, a phenomenon that appeared since the introduction of computer programming, is the change of a software system, over the years and releases, from its initial formation to the point it is withdrawn (is no longer used or surpassed by competitive software). This process requires ongoing maintenance that is remarkably difficult, complicated and time consuming. Software maintenance amounts up to 60% of the resources spent on building an operating a software system [Pres00] and thus, it is of utmost importance for a system's life-cycle. To minimize this effort, developers must comprehend the factors that make software evolve and take active measures to prevent software decay.

In the late seventies, Lehman and his colleagues introduced [BeLe76], a set of rules, also known as the *Laws on Software Evolution*, that govern software evolution. Their findings, that were reviewed and enhanced for nearly 40 years [Lehm96, Leh+97, LeRP98, RaLe00, LeRa01, LeRa06], have been based on their observations upon proprietary and closed source software, not accessible to the public. Those considerations have, since then, given an insight to managers, software developers and software evolution scholars, as to *what* and *why* evolve in the lifetime of a software system.

Being at the very core of most software, databases are also subject to evolution. Every change in the schema of a database implies a severe impact on the surrounding applications. A removal of a relation might cause a complete failure to a query that uses this relation, thus

leading to a syntactic correctness issue. On the other hand, an addition of an attribute that contains important information may be ignored by applications that are not aware of the change, therefore causing a semantic validity problem. Hence, every modification, regardless of its size or type, in the "logical" schema of the database leads to several side effects that affect to both application developers (syntactic) and end users (semantic).

In a similar way to software evolution, we need a set of "laws" or leads as to how evolution takes place in the world of databases. Such rules might give a better insight to database administrators and developers in data-centric applications as to what and how changes in the lifetime of a database schema. Some studies have been previously conducted on the evolution of database schema [Sjob93, PaVa12 and CMDZ13]. Those studies, however, focus on the statistical properties of the evolution of the studied databases and fail to provide detailed data on the actual events of evolution, and do not provide any insight in terms of patterns on the validity of the Lehman laws. To address this gap in literature, we perform a large scale study for the schema evolution of databases that are parts of larger open source projects that are publicly available through open source repositories. Previous studies have given indications on whether Lehman laws apply on open source software [RLWC08, XiCN09] – however not for databases.

The goal of our study is to assess the applicability of Lehman's laws of software evolution to databases in open-source software. The applicability of these laws on databases has not been studied so far, thus making this thesis the first attempt towards a better understanding of the evolution process of the database schema along with the evolution of its accompanied software. We have performed a thorough study on the evolution of several properties of eight databases that we have extracted using our homebrew software tool, Hecate [Skou10]. Such properties include size, growth, and amount of change per version, in terms of both relations and attributes. On the grounds of these observations, we report on the validity of each law.

Our contributions can be listed as follows:

- We perform the first large –scale study of schema evolution in the related literature. We study the evolution of the logical schema of eight databases, publicly available in open-source software.

- We have completed the collection, cleansing and processing of the versions of the database schema for the eight cases that we have studied. We have extracted both the schemata and the differences of database versions and we have come up with the respective datasets that can serve as a foundation for future analyses by the research community.
- We report on measures like the schema size (in terms of relations and attributes), the growth, the growth rate, the breakdown of changes in several categories (additions, removals and updates of attributes, relations and constraints) over both time and version of the eight schemata.
- We study the laws of software evolution in depth, concerning their applicability to the case of databases in open-source software. Due to the high level of abstraction in the wording of the laws and their (meta-) evolution over time, we indulge in a thorough presentation of the intuition behind the wording of the laws. We study how the laws are "translated" for the case of databases. Moreover, we restructure the laws in order to clearly demonstrate their essence.
- We use concrete metrics to assess each law and report on its applicability for the case of schema evolution. We avoid overgeneralizing our findings and present our observations with a focus on measurable properties of the studies databases.

Roadmap. The structure of this thesis is as follows. In Chapter 2, we present related work on software evolution in general, a more specific approach on open source software and three case studies associated with database schema evolution. In Chapter 3, we introduce eight databases that are part of open source software along with sets of measures that illustrate their evolution. In Chapter 4, we dive in Lehman laws on software evolution, and present their definition, the associated metrics needed to evaluate each law and our findings on the applicability those laws to open source software. Chapter 5 summarizes our findings on schema evolution for open source software and concludes with issues that need further research.

CHAPTER 2. RELATED WORK

One of the first thorough investigations on measuring schema evolution and its impact on surrounding applications was made by Sjøberg D. The author of this work has created a tool [Sjøb91] that stood on top of a health management system (HMS) that used a relational database and monitored its evolution for a time frame of 18 months. The results of this study were introduced in [Sjøb93]. The structure, of the one database schema that was examined, had more than doubled its elements during its lifetime (139% for relations and 274% for attributes). The consequences of this evolution, was significantly large (a cumulative 45% of all the names that were used in the queries had to be deleted or inserted). Such great impact, as the author suggests, confirms the need of change management tools that can propagate the change of the schema to the affected queries without making the system unavailable.

Another case study, on a real life open source web information system, is [CMTZ08]. The authors made an analysis on the database backend of MediaWiki, the software that powers Wikipedia. A similar analysis was also made on MediaWiki in [Skou10] with a different tool (Hecate) but having the same results and is again investigated and presented in this Thesis. [CMHZ08] introduces the tool that was used in [CMTZ08], the PRISM workbench. PRISM, a change management tool, provides a language of Schema Modification Operations (SMO) to express schema changes. An SMO is a function that takes as input an initial schema of a database and produces as output a modified version of that schema. The set of those functions is as follows:

- CREATE TABLE
- DROP TABLE
- RENAME TABLE
- COPY TABLE
- MERGE TABLE

- PARTITION TABLE
- DECOMPOSE TABLE
- JOIN TABLE
- ADD COLUMN
- DROP COLUMN
- RENAME COLUMN

Those SMOs are then used to represent the evolution of the schema. Moreover, their nature allows the definition of a sequence of statements implementing the operation semantics in SQL, thus automating query rewriting for the queries that were affected by the schema evolution. PRISM, which was written in Java, was recently reengineered to PRISM++ [CMDZ13], with a faster rewriting engine that can handle updates, integrity constrains and queries with functions. PRISM/PRISM++, that was used to study the evolution of numerous database schemas (most of them are also presented in this Thesis), is a rather forward approach on schema evolution in contrast to the work by Sjøberg, who attempted a "reverse engineering" on schema evolution.

Finally, recent work in [PVSV12] has been made in terms of data warehouses, rather than traditional databases. The evolution of data warehouses has immediate impact on the Extract-Transform-Load (ETL) flows that are attached to them. 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. They asses the quality of different ETL designs with respect to their maintainability and resilience to changes in the data warehouse. Their findings indicate schema size and module complexity (the number of internal edges in the graph representation of an ETL) as important factors for the vulnerability if a system.

CHAPTER 3. DATASETS AND CHARACTERISTIC FACTS

In this chapter we introduce eight datasets from Open-Source software that we collected, sanitized, and studied using our open source SQL diff tool, Hecate [Skou10]. Those datasets come from a wide range of applications such as Content Management Systems (CMS's), Web Forums, Web Stores, Image Galleries as well as Medical and Scientific storages. Opening our discussion, we present (a) our experimental method, (b) the data model for the database schema versions and their differences and (c) algorithm and the differentiate algorithm, based on Sort-Merge Join algorithm, that our tool, Hecate, uses to identify changes between schema versions. Then we present each dataset with its measures. The original collection of the datasets was made by Carlo Curino.[1].

3.1. Experimental Method

For each dataset we gathered as many schema versions (DDL files) as we could from their public source code repositories (cvs, svn, git). The links of those repositories can be found in Table 3.1. 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 commits of the database at the time of the trunk or master branch.
- We ignored all other branches of the project.
- We ignore commits of other modules of the project that did not affect the database.

[1] http://data.schemaevolution.org

The files were collected during June 2013. For all of the projects, except ATLAS Trigger, we focused on their release for MySQL RDBMS. For the ATLAS Trigger database we have the Oracle version as no MySQL version was developed Those files where then renamed with their filenames matching to the date (in standard UNIX time) the commit was made. The files were then processed by sequential pairs from our tool, Hecate, to give us (a) the differences between two subsequent commits and (b) the measures we needed to conduct this study. Hecate, which is written in Java, parses the DDL files and creates the appropriate objects that represent the schema structure of the database. Those objects are then compared and a list of measures is extracted about the change that took place between them.

3.1.1. Hecate SQL Model and Mapping

Hecate parses the DDL files and identifies CREATE and ALTER commands with the help of ANTLR v4[1]. The parser is quite error resilient as all of the datasets contained syntactic and typing errors and almost none of them followed the naming restrictions of MySQL or the general SQL query specifications. Such behavior is most likely due to the high activity of the main development branch, that we targeted, which, in many cases, contained some erratic commits that were later corrected. We are interested in the logical schema, thus we ignore changes to the physical schema (indexes, storage engines, locales and other DBMS options). After parsing the files, Hecate creates sorted lists of relations and the attributes they contain as well as foreign key constrains for the relations.

3.1.2. Differentiating DDLs

For the items to be sorted, we compare the alphanumeric strings of the names of the items to indicate their ranking on the list. This means that all schema elements are defined by their name and, because no renaming detection was implemented, a possible rename is processed as a deletion and an insertion. Exploiting the sorted lists of Relations and Attributes, Hecate uses a Sort-Merge Join algorithm to determine whether the two items in question are matching or something was inserted or deleted from the schema and marks them appropriately. Attributes are marked as altered if they exist in both versions and their type or participation in theirs relation primary key changed. Relations are marked as altered if they

[1] http://www.antlr.org/

exist in both versions and their contents had a change (attributes inserted/deleted/altered). Summarizing, the changes that we detect are the following:

- Attributes inserted
- Attributes deleted
- Relations inserted
- Relations deleted.
- Attribute type change
- Attribute participation in a Relations Primary key change

Those changes can be massively detected in an automated way. There is no ambiguous or estimation to what has happened compared, for example, detecting renames where we estimate the existence of a renaming based on name similarity and data type. Thus, the reported facts are accurate deltas, automatically extracted from the repositories.

Before proceeding, however, we need to establish a clear terminology, along with a set of definitions for the measured properties of the datasets that we have analyzed. This is the topic of the following subsection.

3.2. Terminology, Definitions and Notations

In this subsection, we start by giving the definitions and terminology for the concepts of dataset, version, transition, and revision. Then, we move on to define the measurable properties, or measures, that pertain to each of them.

3.2.1. Definitions of main concepts

Schema Version, or simply, Version: A snapshot of the database schema, committed to the public repository that hosts the different versions of the system. To facilitate our deliberations, we assign an artificial version id to each version, in the form of an auto-incrementing integer with step one. Thus, version ID's are isomorphic to a contiguous subset of the set of positive integers. Whenever possible, we also assign a timestamp to the version, which is the commit time to the public repository.

Synonymous term: Commit, Revision.

Dataset: A sequence of versions, respecting the order by which they appear in the repository that hosts the project to which the database belongs. In our case, we have monitored only the versions committed to the trunk (master development branch) of the project to which the database belongs.

Transition: The fact that the database schema has been migrated from version v_i to version v_j, $i < j$. We refer to v_i as the source version of the transition and to v_j as the target version of the transition. We denote such a transition by an arrow from the source towards the target of the transition, e.g., $v_i \rightarrow v_j$. Throughout all our deliberations, we employ the term old to refer to properties of the source version and the term new to refer to properties of the target version of a transition.

Revision: A transition between two sequential versions, i.e., from version v_i to version v_{i+1}. Each transition incurs a set of differences to the database schema. We measure the alteration of tables and attributes in a manner that will be clearly defined right away. To simplify expressions, we frequently use the term "version" instead of "transition to leads to this version". So, for example, if we say the "new number of relations for version v_i", we refer to the number of tables of v_i, after a transition has taken place. Unless otherwise specified, such a transition is a revision, i.e., a transition $v_{i-1} \rightarrow v_i$.

3.2.2. Definitions of measured properties for transitions and datasets, quantifying change

We classify the measured properties of the evolution of a database schema – to which we refer as measures – into three categories.

A. Properties of a single transition.

We remind the reader that, unless otherwise specified, in the rest of our deliberations, the studied transitions are revisions. However, in the current discussion, the definitions are given for transitions in the general case.

i. Measures pertaining to relations.

Num Old Tables: The number of Tables of the first (oldest) schema version of a transition.

Num New Tables: The number of Tables of the second (newest) schema version of a transition.

Table Insertions: The number of Tables inserted during a transition.

Table Deletions: The number of Tables removed during a transition.

Table Modifications: The number of Tables that coexist between two schema versions of the transition and have changes such as attributes inserted, attributes removed, attributes having changed type, and attributes becoming primary keys for the Table.

ii. Measures pertaining to attributes.

Num Old Attributes: The number of Attributes of the first (oldest) schema version of a transition.

Num New Attributes: The number of Attributes of the second (newest) schema version of a transition.

Attribute Insertions: The number of Attributes inserted in Tables that exist in the source (oldest) version of a transition.

Attribute Inserted at Table Formation: The cumulative number of Attributes of all the new Tables inserted in the database schema during a transition.

Attribute Deletions: The number of Attributes removed from Tables that reside in both versions of the transition.

Attribute Deletions at Table Removal: The cumulative number of Attributes of all the Tables that were removed from the database schema during a transition (i.e., cease to exist in the new version of the transition, although present in the old version).

Attribute Type Alternations: The number of Attributes that changed type during a transition.

Attribute Key Alternations: The number of Attributes that reversed their status concerning their participation to the primary key of a relation – specifically, the number of attributes that either became primary keys in the new version (while they were not in the old version) or stopped being primary keys in the new version while being primary keys in the old version.

Attribute Alternations: The sum of Attribute Type Alterations and Attribute Key Alterations.

Schema Growth: The difference between the new and old Tables of a transition, i.e., New Tables – Old Tables (attn: not the absolute, but the actual value of the subtraction).

Schema growth ratio: Growth divided by the size of the older version. (i.e., the formula is Schema Growth / Num Old Tables)

iii. Cumulative measures for a transition

Transition Change Breakdown: a tuple of measures for the transition under discussion, including Table Insertions, Table Deletions, Attribute Insertions, Attribute Deletions, Attribute Inserted at Table Formation, Attribute Deletions at Table Removal, Attribute Alternations.

Transition Changes: The sum of Table Insertions + Table Deletions + Attribute Insertions + Attribute Deletions + Attribute Inserted at Table Formation + Attribute Deletions at Table Removal + Attribute Alternations for a transition.

B. Timeline measures for entire datasets

Once we have reviewed the measures for individual transitions, we can extend them to timeline measures – i.e., the monitoring of these measures for a sequence of schema versions. Whenever we append the term "timeline" to the aforementioned measures, we refer to a sequence of such measurements, sorted by their version id, assessing the investigated measure

for each version id. Unless otherwise specified, we refer to entire datasets as sequences of schema versions; again, transitions are revisions. To give a few examples:

Num New Tables Timeline: a sequence of measurements, each measuring the tables of the new version of a revision. The timeline practically gives the schema size for each version.

Attribute Deletions Timeline: a sequence of measurements, each measuring the number of attributes deleted from surviving tables of a transition.

Change Breakdown Timeline: a sequence of tuples, one per transition, with its change breakdown.

C. Cumulative properties of a dataset.

Num Versions: The total number of versions for a dataset.
Lifetime: The time interval (in standard UNIX time) between the first and last commit for a dataset.

Total Num Changes: The sum of Changes of all revisions.

Complexity Maintenance Rate: The number of evolution-affected Tables over the number of Tables of the original database schema during a transition. The exact formula is (Tables Inserted + Tables Modified) / Old Tables. Observe that maintenance rate is different from growth ratio, as it includes the modified tables in its formula (as opposed to the growth ratio).

Changes per Day: The number of all revisions divided by the schemas lifetime.

3.3. Presentation of Measures

Following we provide a brief description of each dataset (in alphabetical order) and some macroscopic figures for an insight about those measures. Those figures are presented in three groups:

Events over time: We start with a set of figures that depict the evolution of events over time and specifically, the number of events over time, the schema size (no. of tables and no. of attributes). The first figure of this group shows the accumulative changes made on the database schema over time. Due to the restricted size of the figures, in order to fit in those pages, compared to large lifetimes of the databases, some changes overlap on the figure. Moreover, there are occasions where too many commits are applied within a short period of time. For a more detailed (with each type of change) figure look at the next group of figures. The two following figures show the number of relations and attributes over time. Areas where changes seem to be large but the schema size remains the same indicate that same number of schema elements where added and deleted (maybe some of them renamed) or the changes are alterations of the schema (attributes change type or relations change primary keys). A combined figure of those figures can be found on the third group.

Analysis of Changes over Time: Once we have presented the evolution of events over time, we are now ready to drill in this information and study the breakdown of these events. We repeat the first figure with more detail on the type of change over time. We use three colors to indicate changes in this figure. Red is for deletions, blue for insertions and orange for alterations. This approach also suffers from overlapped commits. For this reason we present a zoomed out figure with the changes breakdown over version ID. This second figures uses the same palette of colors as the previous with lighter tone for attributes and even more light tone for attributes born with newly inserted relations. Versions that seem to have no changes indicate that something was changed to the database irrelative to the schema structure (i.e. database management system parameter change). Transitions that have dominance of blue or red indicate growth or diminish on schema size respectively.

Further insights: To further probe into this information, we also provide some extra analyses. The first figure is a combination of the relations and changes figures from the first group but with the x axis representing the version ID of the schema. The second one is an intuitive way to represent the schema evolution regarding its size. The x axis holds the total number of relations for the schema while the y axis holds the total number of attributes. Each point represents a version of the schema and the lines connecting those dots, the change between two subsequent schema versions. Because of the general increasing behavior the datasets have most of them (except phpBB) has their initial version on the bottom left of the figure and the

last on top right. Vertical lines indicate that attributes were added while the total number of relations remained the same. Horizontal lines (rare) indicate the reverse. Those assumptions should not be considered with absolute confidence, but rather as indications, because relations or attributes might be inserted or removed thus leaving the schema size the same.

Table 3.1 Collected Datasets

Dataset	Versions	Lifetime	Tables Start	Tables End	Attributes Start	Attributes End	Commits per Day	% commits with change	Repository URL
ATLAS Trigger	84	2 Y, 7 M, 2 D	56	73	709	858	0,089	82%	http://atdaq-sw.cern.ch/cgi-bin/viewcvs-atlas.cgi/offline/Trigger/TrigConfiguration/TrigDb/share/sql/combined_schema.sql
BioSQL	46	10 Y, 6 M, 19 D	21	28	74	129	0,012	63%	https://github.com/biosql/biosql/blob/master/sql/biosqldb-mysql.sql
Coppermine	117	8 Y, 6 M, 2 D	8	22	87	169	0,038	50%	http://sourceforge.net/p/coppermine/code/8581/tree/trunk/cpg1.5.x/sql/schema.sql
Ensembl	528	13 Y, 3 M, 15 D	17	75	75	486	0,109	60%	http://cvs.sanger.ac.uk/cgi-bin/viewvc.cgi/ensembl/sql/table.sql?root=ensembl&view=log
MediaWiki	322	8 Y, 10 M, 6 D	17	50	100	318	0,100	59%	https://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/maintenance/tables.sql?view=log
OpenCart	164	4 Y, 4 M, 3 D	46	114	292	731	0,104	47%	https://github.com/opencart/opencart/blob/master/upload/install/opencart.sql
phpBB	133	6 Y, 7 M, 10 D	61	65	611	565	0,055	82%	https://github.com/phpbb/phpbb3/blob/develop/phpBB/install/schemas/mysql_41_schema.sql
TYPO3	97	8 Y, 11 M, 0 D	10	23	122	414	0,030	76%	https://git.typo3.org/Packages/TYPO3.CMS.git/history/TYPO3_6-0:/t3lib/stddb/tables.sql

3.4. ATLAS Trigger

ATLAS[1] is a particle physics experiment at the Large Hadron Collider at CERN, the European Organization for Nuclear Research based on Geneva, Switzerland. ATLAS goal is to learn about the basic forces that have shaped our universe since the beginning of time and that will determine its fate. Among the possible unknowns are the origin of mass, extra dimensions of space, microscopic black holes, and evidence for dark matter candidates in the universe. Trigger is one of the software in the ATLAS project responsible of filtering the immense (40 terabytes per second[2]) data collected by the Collider and storing them in its database. As mentioned before, this database uses the Oracle RDBMS. The database schema started with 56 relations and after two and a half years grew to the size of 73 relations. Despite the projects short lifetime, this schema had very active evolution with 82% of changes affecting its design.

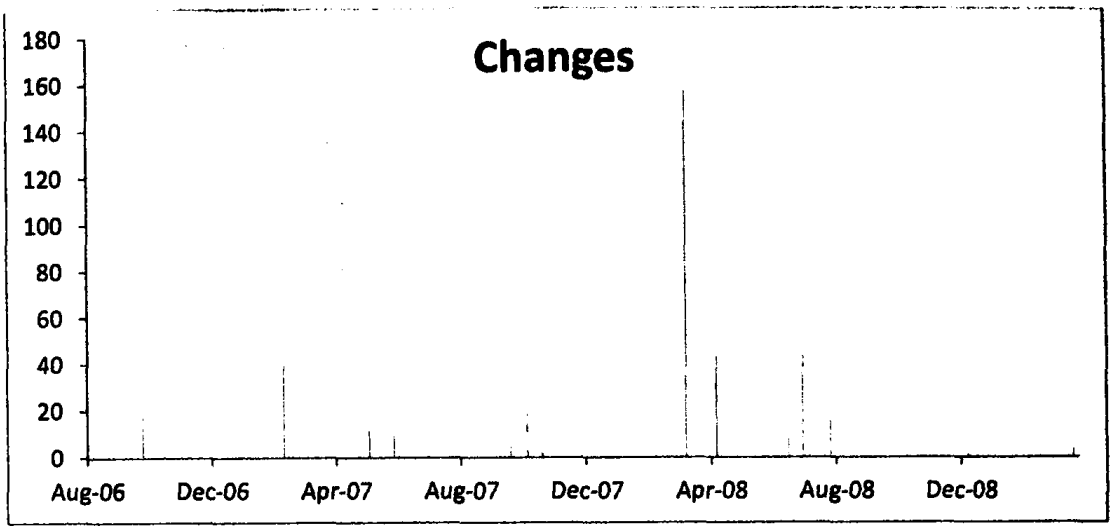
The database schema structure had two major reconstructions, in the two summers of 2007 and 2008 (Figure 3.1). Generally, it looks like summers are high activity areas for the development of this database. The great change around March 2008 has no structure impact which means that the database had some non structural change such as type or key alterations.

After a closer look at Figure 3.2, we can now confirm that large bars from the previous set of figures were alterations on the database schema rather than direct structural change. After a closer look on the above figures, we can see that the database schema has a steady growth with one big drop in relations and attributes just after revision 22. In revisions 19, 32, 33, and 49 we have a large number of relation alterations that affect almost the entire schema.

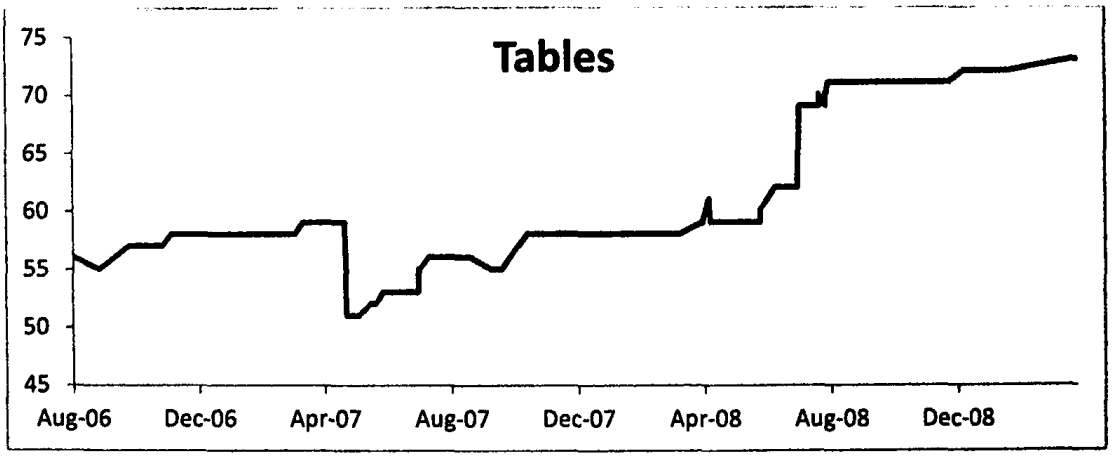
After the major drop in relations after revision 22, as shown in Figure 3.3 the database schema seems to have a more stable grow with a big leap ahead in revision 69. We can see a quite large drop in relations and attributes in the early life of the schema and an almost steady growth afterwards.

---

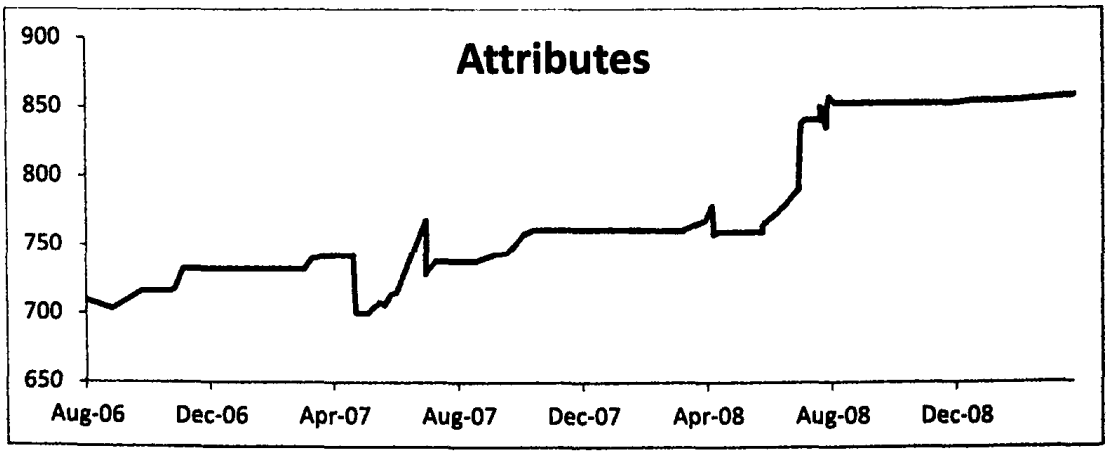
[1] http://atlas.web.cern.ch/Atlas/Collaboration/
[2] http://www.atlas.ch/pdf/atlas_factsheet_4.pdf



a. Change Heartbeat over Time for ATLAS Trigger

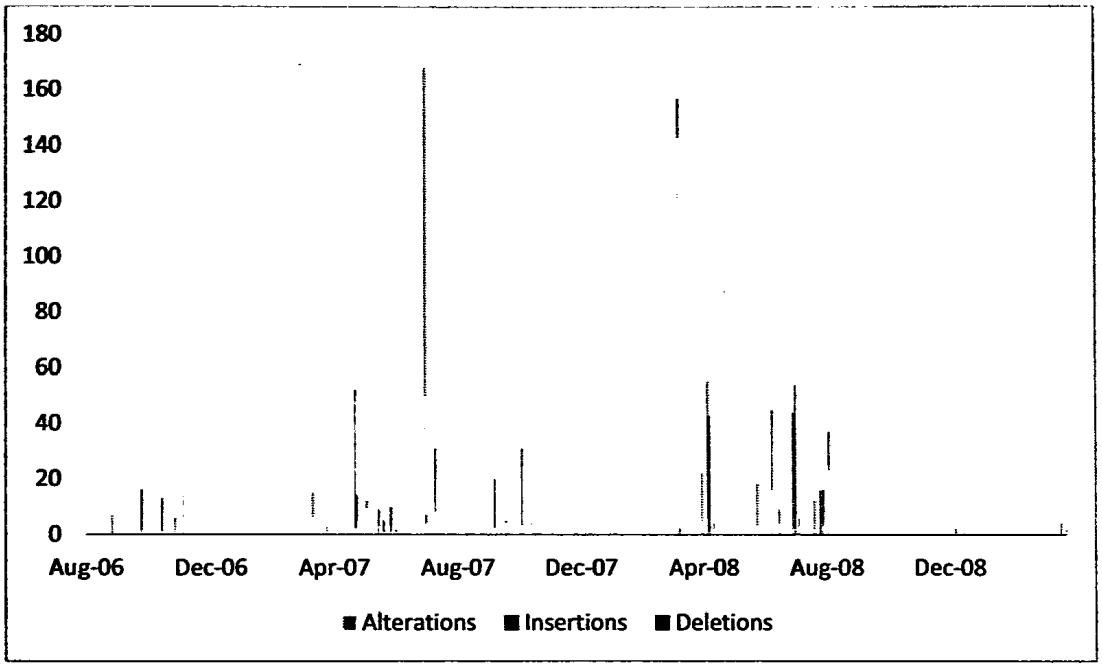


b. Number of Tables for ATLAS Trigger

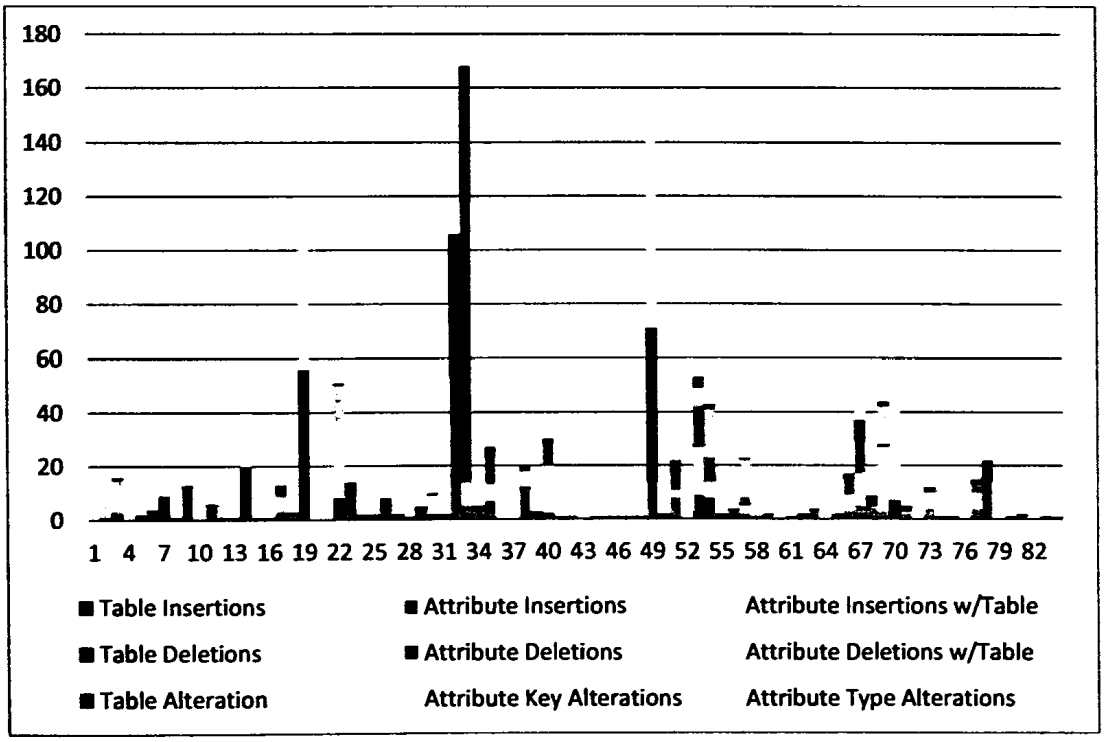


c. Number of Attributes for ATLAS Trigger

Figure 3.1 Events over time for ATLAS Trigger

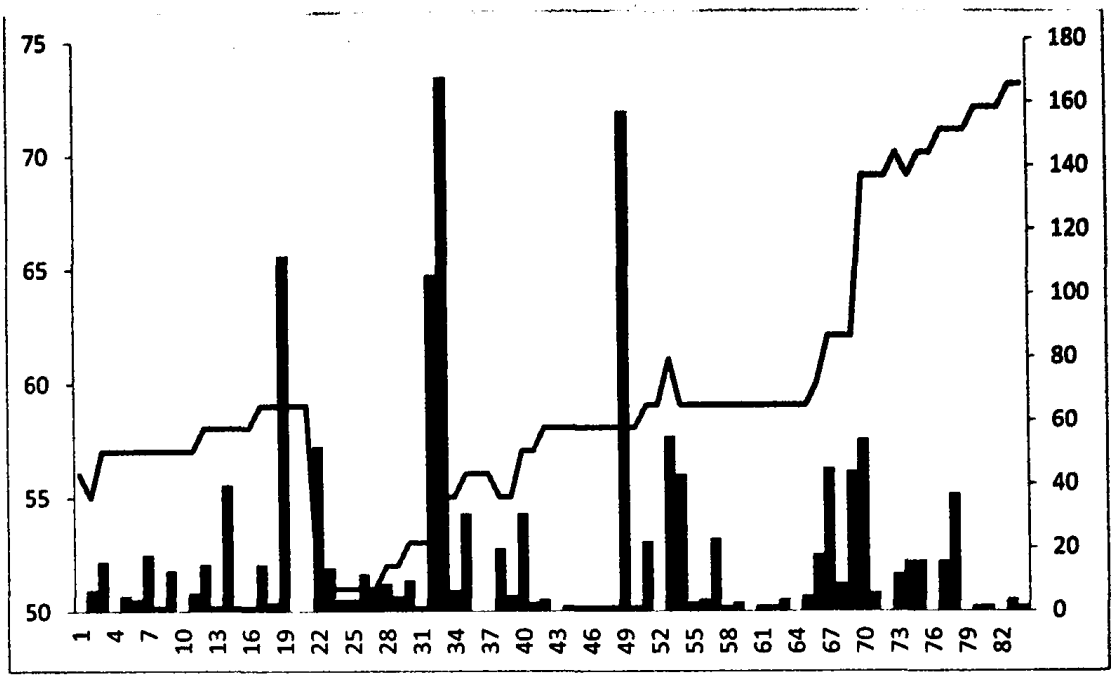


a) Changes Breakdown over Time for ATLAS Trigger

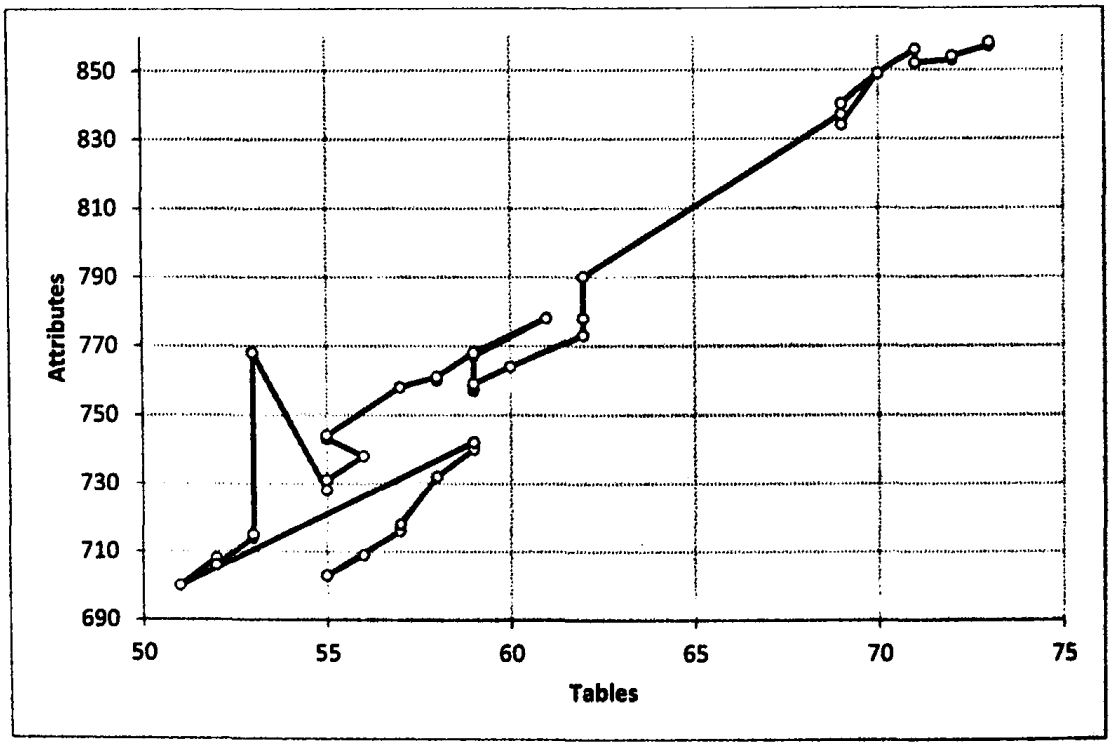


b) Changes Breakdown over Version ID for ATLAS Trigger

Figure 3.2 Analysis of Changes over Time for ATLAS Trigger



a) Changes compared with schema size in relations for ATLAS Trigger



b) Timeline for ATLAS Trigger

Figure 3.3 Further Insights on ATLAS Trigger

3.5. BioSQL

BioSQL[1] 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[2] or Swissport[3]. While in its original incarnation (back in 2001) it was conceived by Ewan Birney as a local relational store for GenBank, the project has since become a collaboration between the Open Bioinformatics Foundation (OBF)[4] projects (those include 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 the Bio* projects. Each Bio* project has a language binding (object-relational mapping, ORM) to BioSQL. Thus Entries stored through an application written in, say, Bioperl could be retrieved by another written in Biojava. The currently supported Relation Database Management Systems (RDBMSs) are MySQL, PostgreSQL, Oracle and most recently SQLite. At the time that this thesis was composed the SQLite solution was only supported by Biopython.

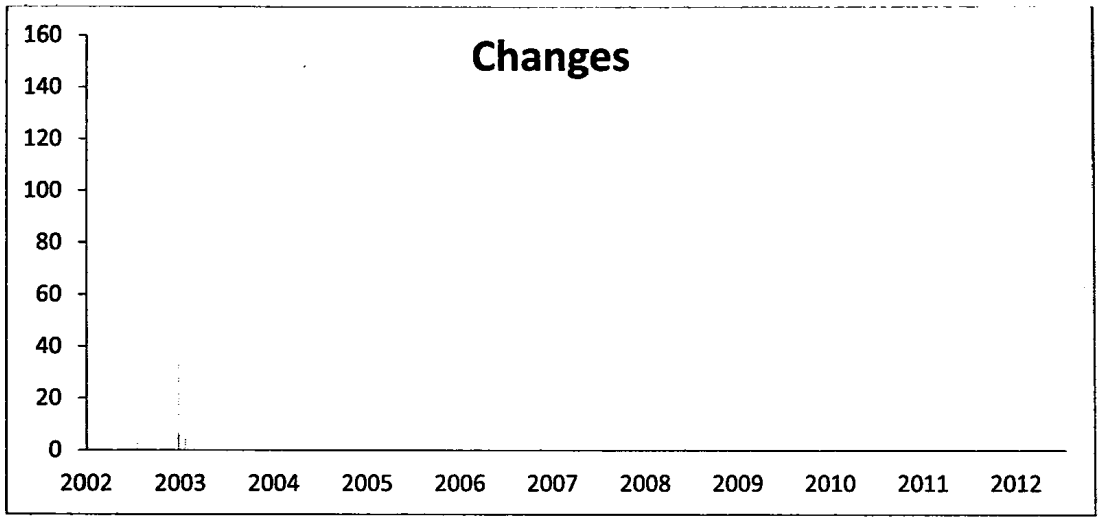
For some reason, currently not known to us, this project was frozen sometime around mid 2005 as we can clearly see in Figure 3.4.a. This happened to be the time that OBF became an organization with formal members and bylaws. In 2012, the board decided to give up their our own incorporation to associate themselves with Software In The Public Interest, Inc.[5], a fiscal sponsorship organization that they felt aligned well with their values and culture. The bylaws underwent a series of changes to pave the way for joining SPI. The changes were approved on Sep 11, 2012, which happens to be the last update to the schema version (Figure 3.4.a). Most recent changes (after 2005) did not affect the schema structure. Most of the activity for this project is concentrated in early 2003, where the schema had a major growth. Especially the number of attributes had a more than 50% increase in their number.

Figure 3.5.a does not give us clear information due to the large gap between the main development period of the database and the last revision in 2012. A more clear view of the activity can be seen on Figure 3.5.b. Revision 3, 10 and 27 have almost the same insertions

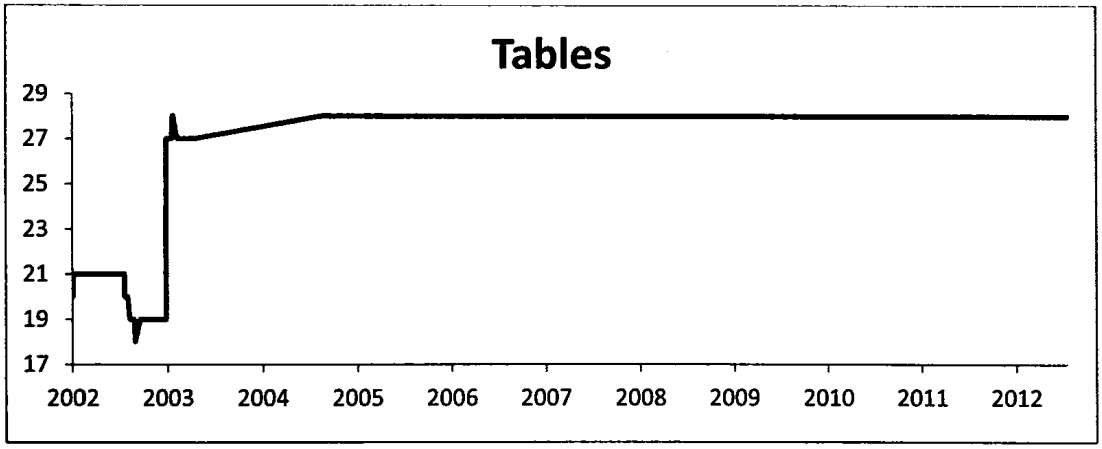
[1] http://www.biosql.org/wiki/Main_Page
[2] http://www.ncbi.nlm.nih.gov/genbank/
[3] http://www.uniprot.org/
[4] http://www.open-bio.org/wiki/Main_Page
[5] http://spi-inc.org/

and deletions indicating a massive renaming and restructuring in the components of the schema. Apart from those periods the schema had minor changes in its structure with a mix of different types of changes, with attribute type alterations occurring only in version 10.

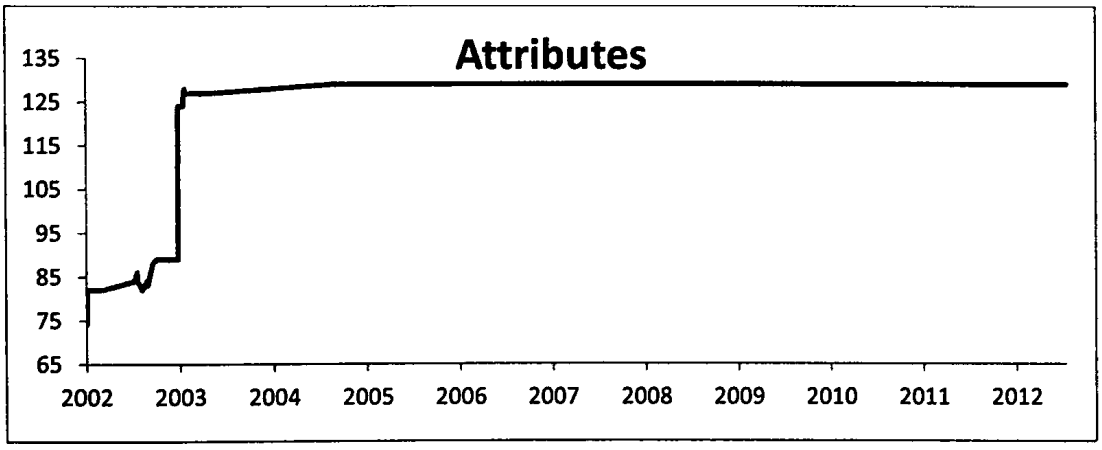
Supporting the previous intuition about revisions 3, 10, and 27 we observe in Figure 3.6 that the schema did not change much in size on those revisions. We also observe the great increase in the number of tables at revision 21. The timeline of BioSQL shows an uneasy first period with the number of both relations and attributes falling until a point where, after a great leap forward, the growth becomes steady for the rest of the schemas lifetime.



a) Changes Heartbeat over Time for BioSQL

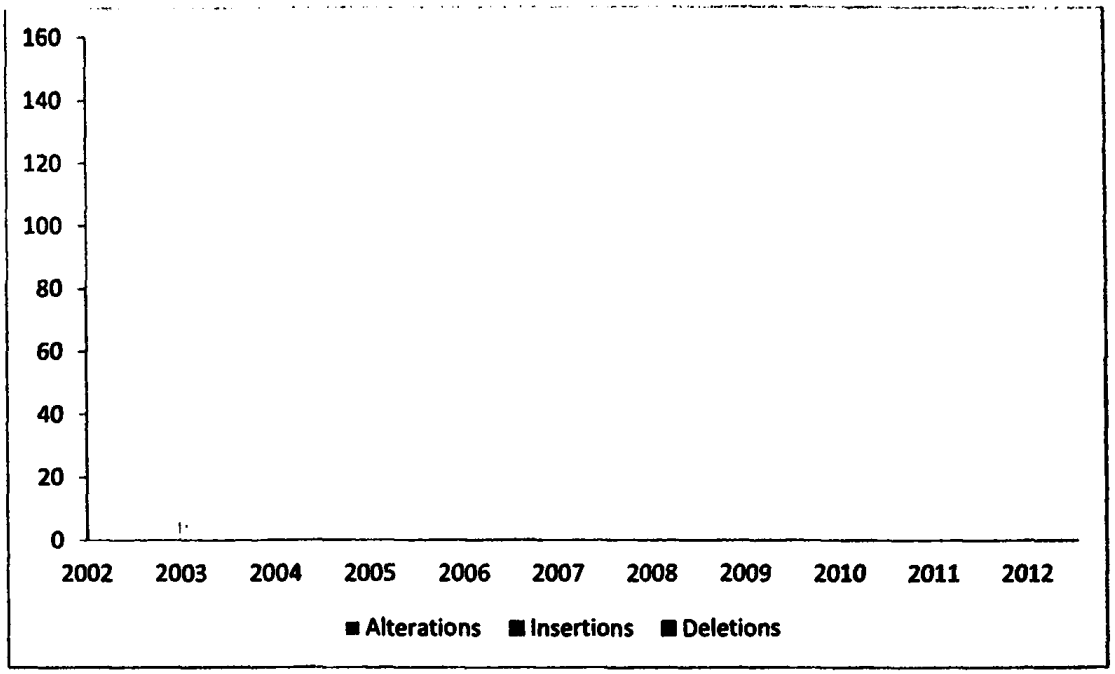


b) Number of Tables for BioSQL

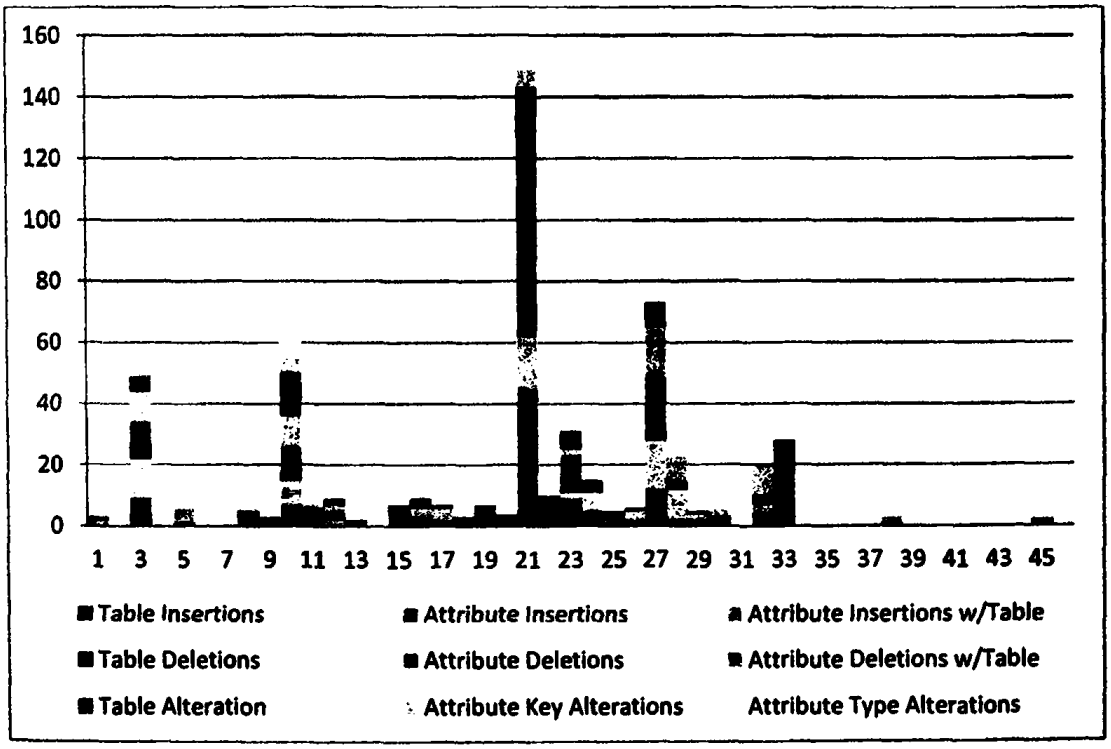


c) Number of Attributes for BioSQL

Figure 3.4 Events over Time for BioSQL

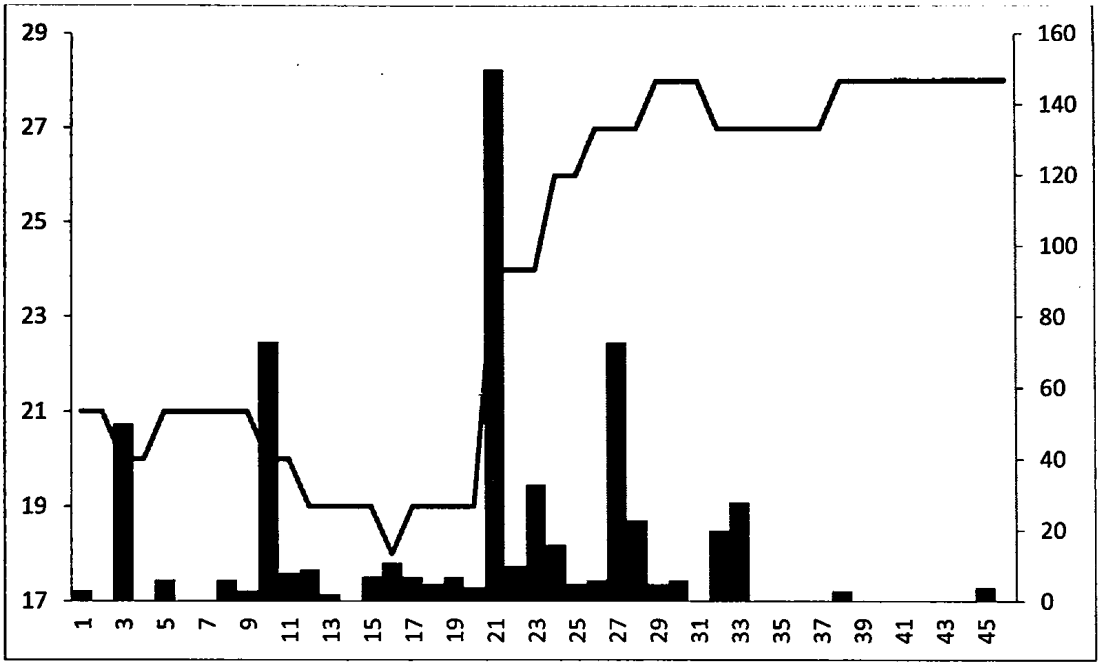


a) Changes Breakdown over Time for BioSQL

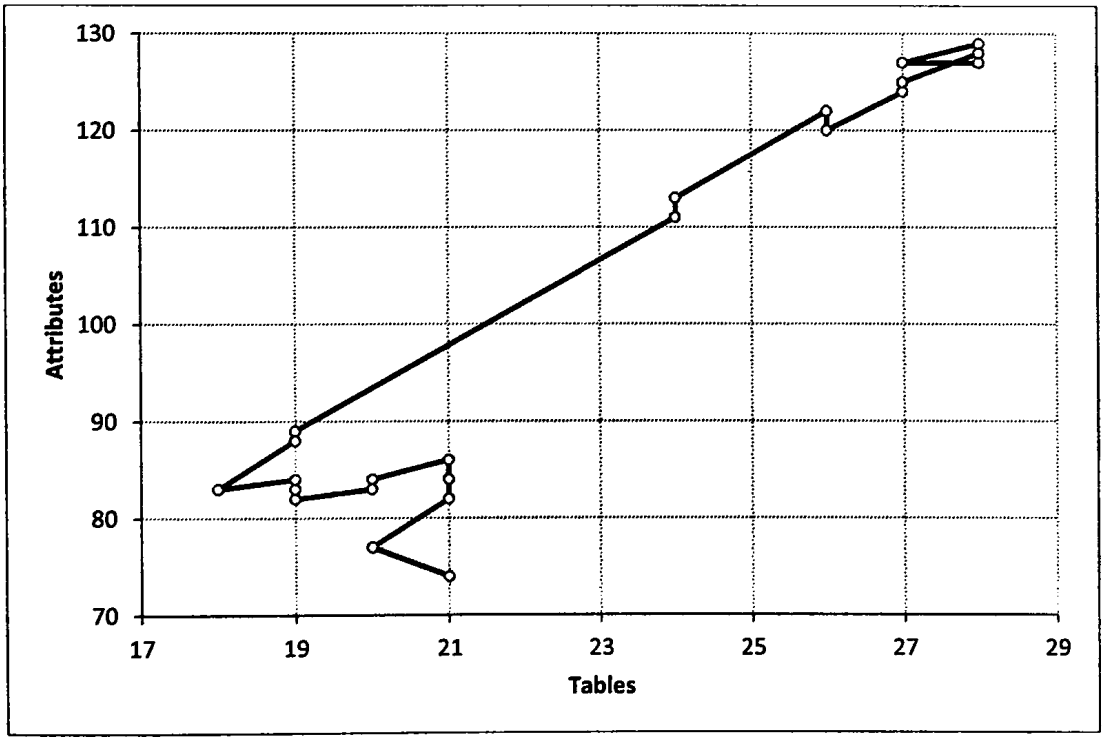


b) Changes Breakdown over Version ID for BioSQL

Figure 3.5 Analysis of Changes over Time for BioSQL



a) Changes compared with schema size in relations for BioSQL



b) Timeline for BioSQL

Figure 3.6 Further insights for BioSQL

3.6. Coppermine Photo Gallery

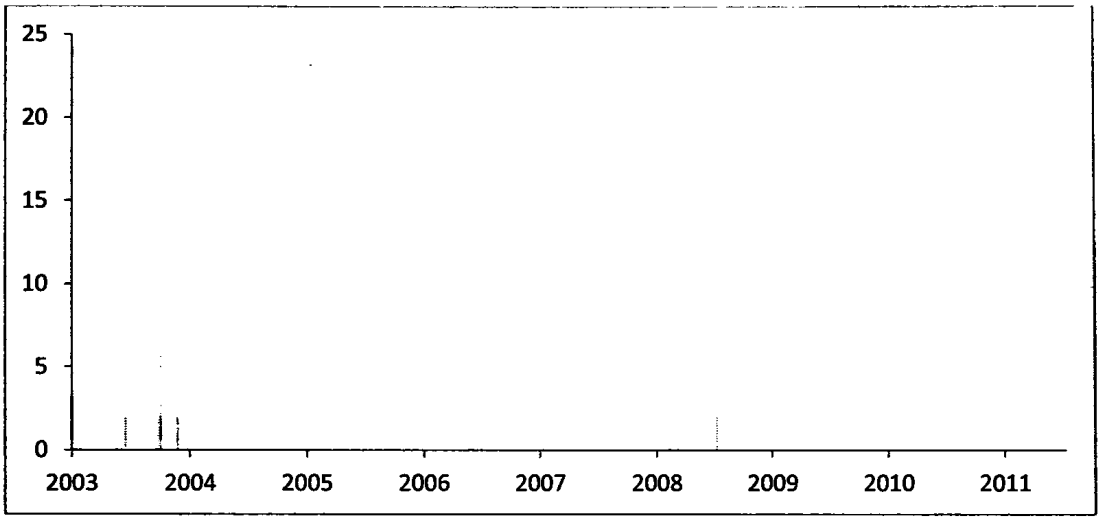
Coppermine[1] is a photo gallery web application. Its features include the arranging pictures in categories and albums, full multimedia support, option for private galleries, automatic thumbnail creation, multi-language interface with automatic detection of language preferences. Installation requires PHP, MySQL, and ImageMagick or the GD Graphics Library, and works with most web server software such as Apache. Coppermine is free software and is being released under the GNU GPL license.

The evolution of this database schema looks front-loaded with many changes early on and fewer and smaller in effect changes later both for relations and attributes (Figure 3.7). From mid 2004 and until early 2006, the schema size remains stable in terms of relations and a minor growth in attributes, with the same behavior from mid 2008 an later on. We observe two periods with a drop in the number of relations as well as with attributes in early 2004 and mid 2008 with attributes following with a drop in their numbers as well. A third drop in attributes in mid 2004 does not seem to affect the number of tables which remain stable while some attributes are inserted until mid 2006.

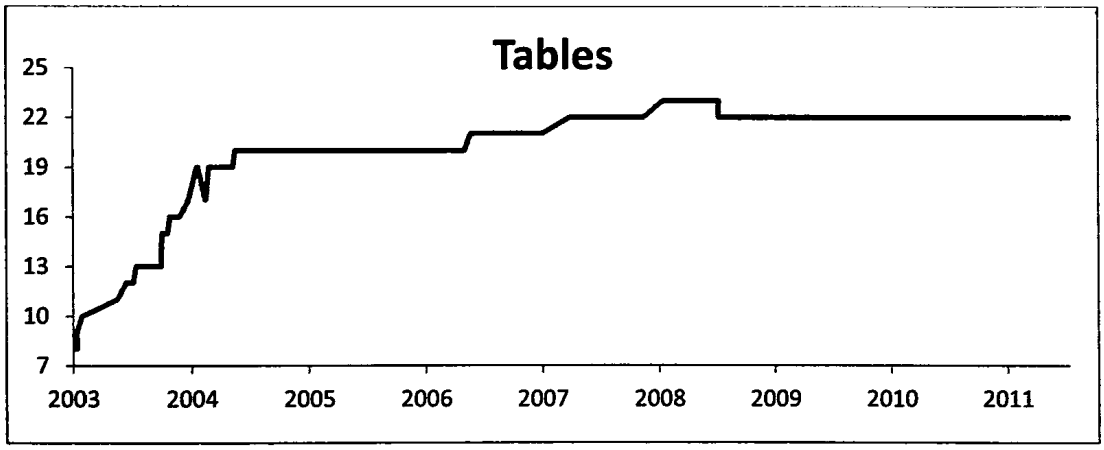
In contrast with the two previous databases we fail to see equal insertions and deletions on the same revision (Figure 3.8). Another interesting point is the fact that, except revision 38, all modifications on the schema affected only one table at a time, which is indicated by the small dark orange bar on top of each column. Moreover, there are almost no alterations on the schemas attributes.

In Figure 3.9.a, we observe in more detail the fact that changes were becoming more frequent until the point (version 30) where the schema doubles in size since its beginning. Figure 3.9.b shows an almost linear growth in schema when both relations and attributes are taken into account, with the exception of the two drops in the number of relations, as mentioned before, from 19 to 17 (with attributes also diminishing in size) and from 32 to 22 (again with attributes dropping). The activity of the schema when relations reach 20 is due to the drop in the number of attributes and their slow and steady recovery that we stated previously.

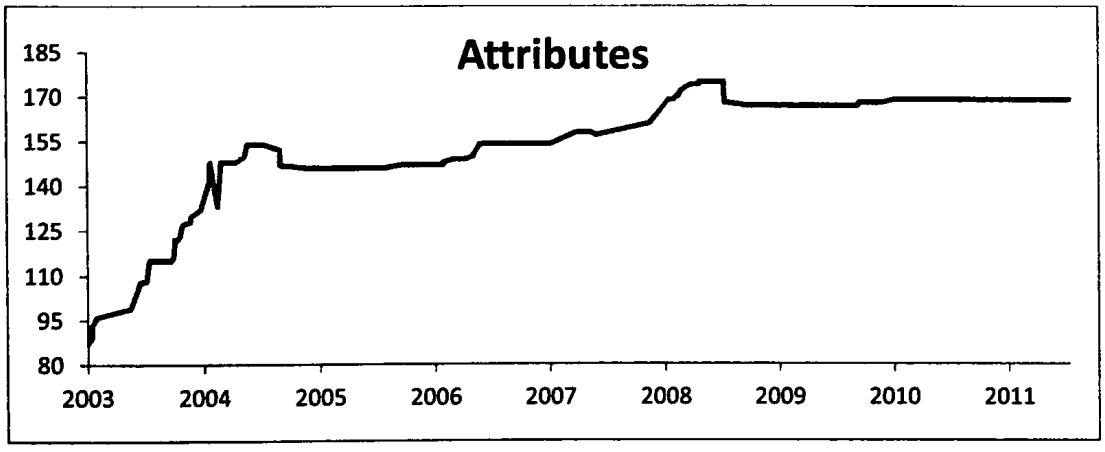
[1] http://coppermine-gallery.net/



a) Cumulative Changes over Time for Coppermine

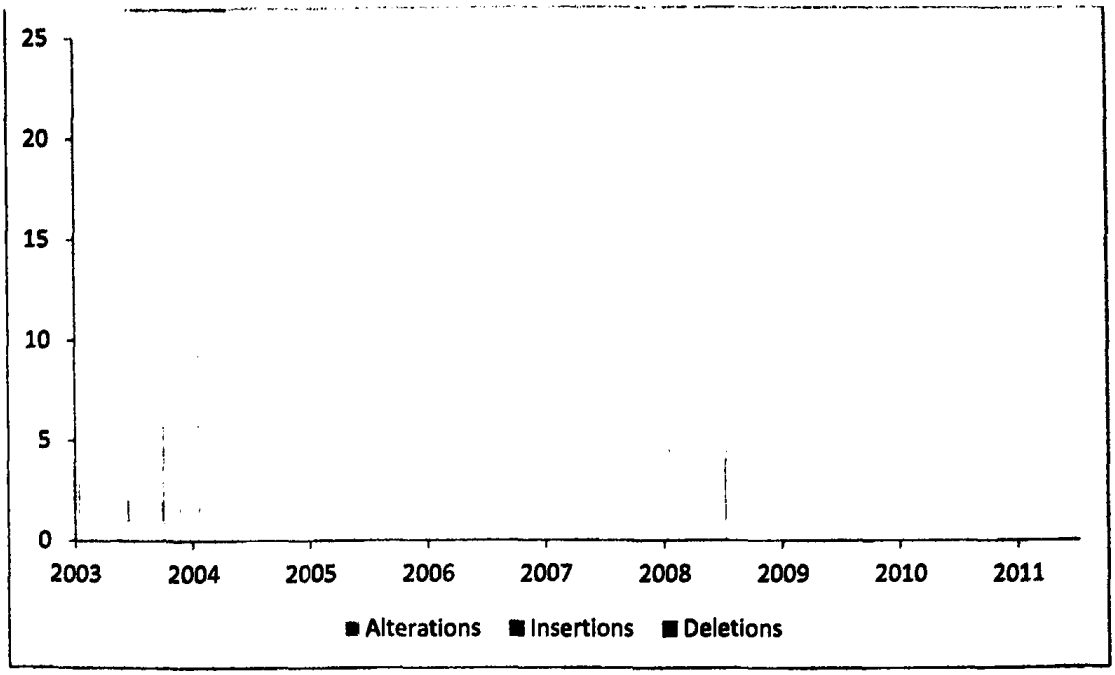


b) Number of Tables for Coppermine

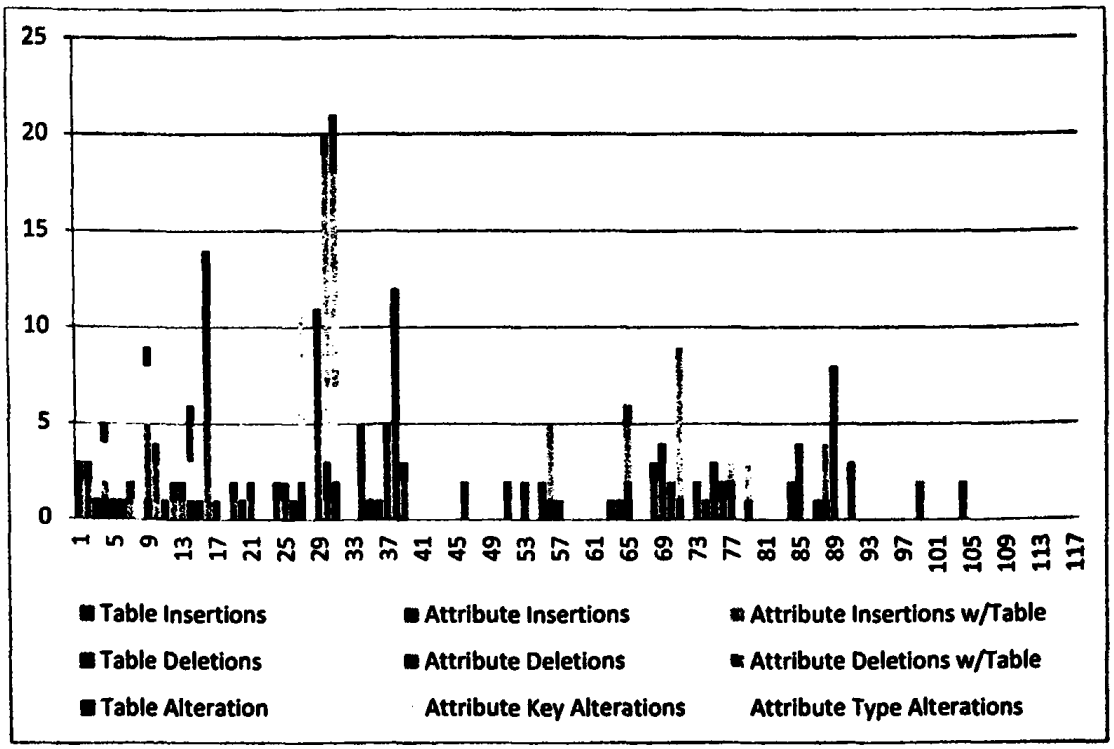


c) Number of Attributes for Coppermine

Figure 3.7 Events over time for Coppermine

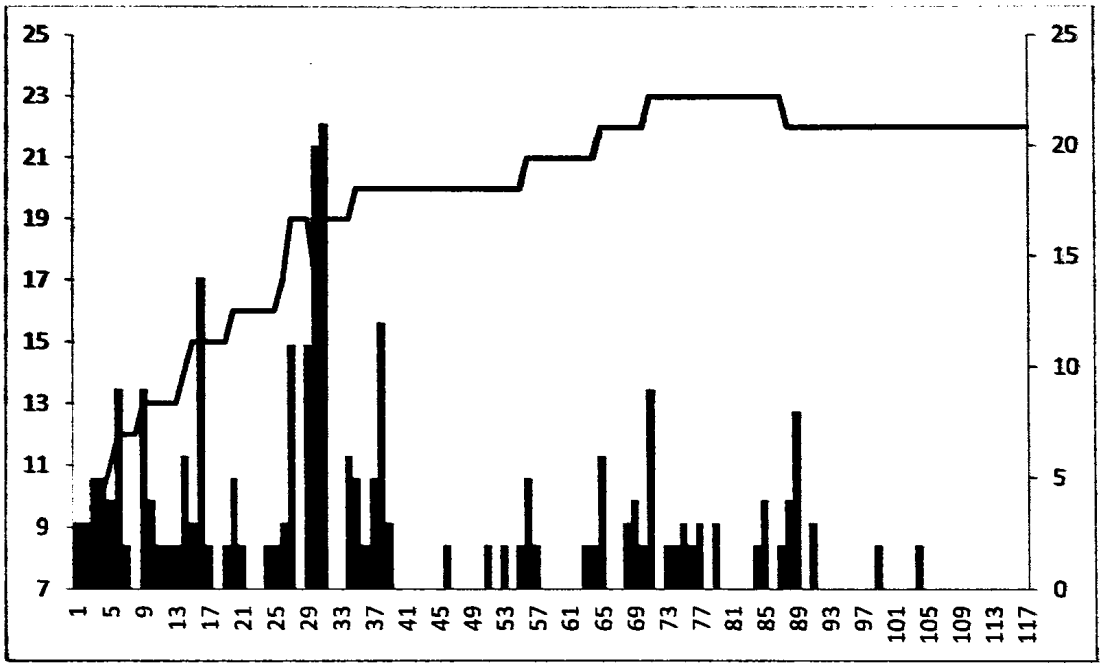


a) Changes Breakdown over Time for Coppermine

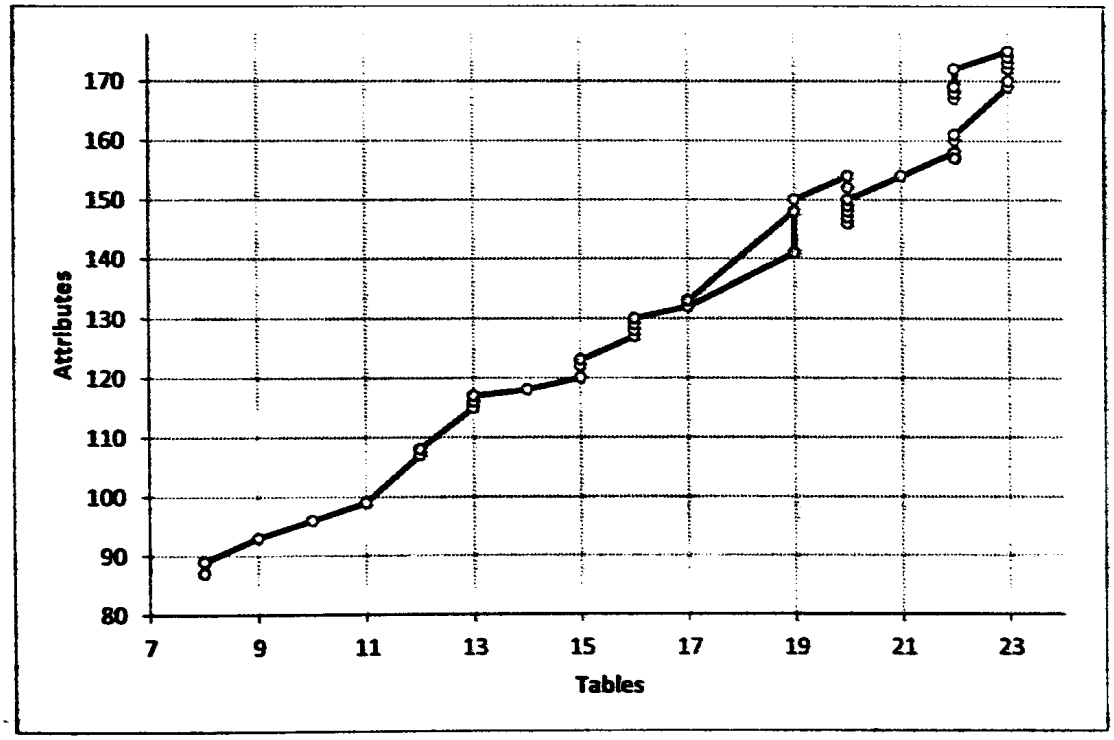


b) Changes Breakdown over Version ID for Coppermine

Figure 3.8 Analysis of Changes over Time for Coppermine



a) Changes compared with schema size for Coppermine



b) Timeline for Coppermine

Figure 3.9 Further insights for Coppermine

3.7. Ensembl

Ensembl is a joint scientific project between the European Bioinformatics Institute (EBI)[1] and the Wellcome Trust Sanger Institute (WTSI)[2], which was launched in 1999 in response to the imminent completion of the Human Genome Project[3]. Even at that early stage it was clear that manual annotation of three billion base pairs of sequence would not be able to offer researchers timely access to the latest data. The goal of Ensembl was therefore to automatically annotate 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. Ensembl provides a genome browser that acts as a single point of access to annotated genomes for mainly vertebrate species. Information such as gene sequence, splice variants and further annotation can be retrieved at the genome, gene and protein level. This includes information on protein domains, genetic variation, homology, syntenic regions and regulatory elements. Coupled with analyses such as whole genome alignments and effects of sequence variation on protein, this powerful tool aims to describe a gene or genomic region in detail.

Due to the large number of revisions and changes in the Ensembl database we present the two last groups of figures in landscape. The y axis in the changes figures has been abridged to 200 changes for a better view of the small bars at the bottom of the figures. Numbers at the top of the bars indicate that those values exceed the 200 restriction and are the actual size of the change.

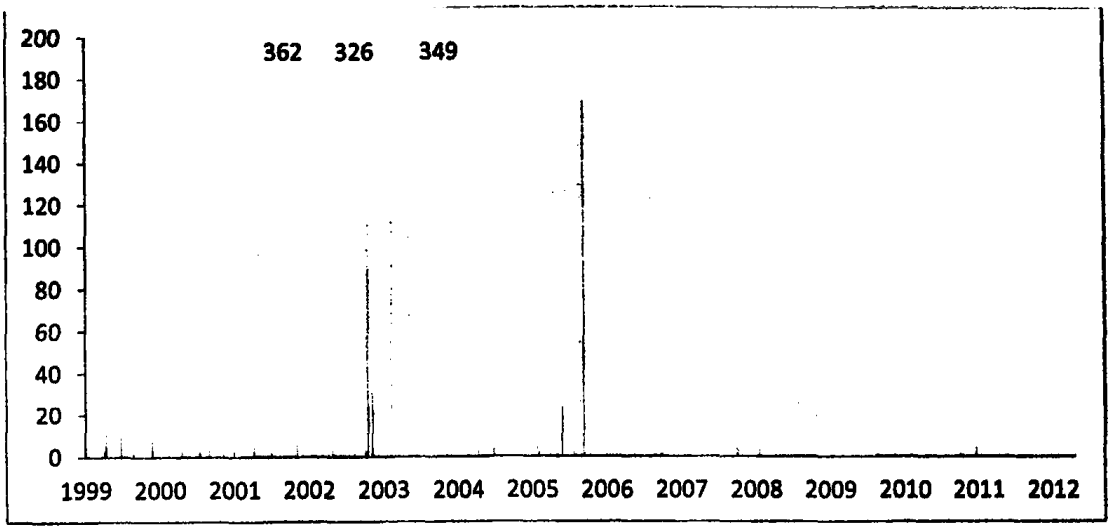
The time data from this project (Figure 3.10) resemble those of Copermine with more intensive periods early on but with much more activity and for a longer period of time. Just like Coppermine we can see a small drop in 2007 in both relations and attributes just before a long "quite" period with few changes. Another remarkable observation is the fact that the schema grew more than triple in size since its beginning, until 2006, when the activity seems to drop with more scarce changes having less impact in the schema size.

---

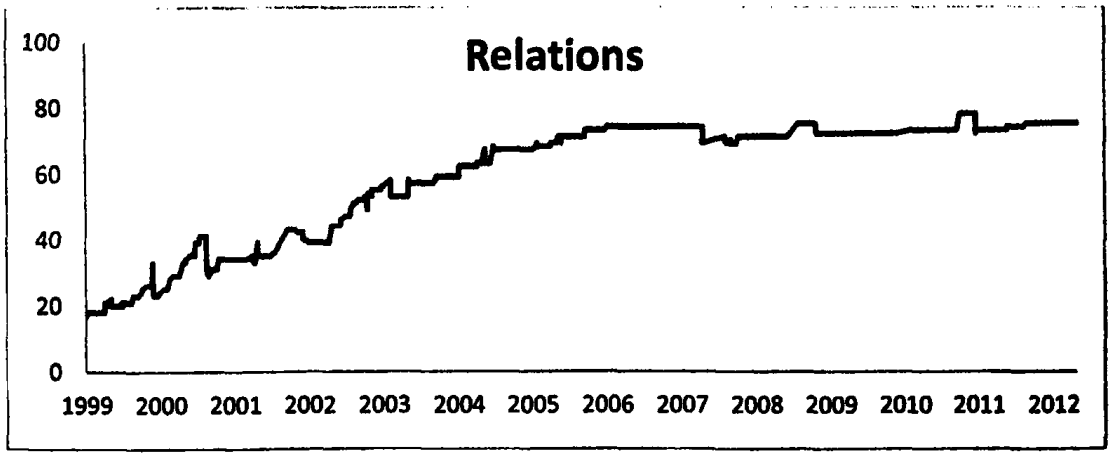
[1] https://www.ebi.ac.uk/
[2] https://www.sanger.ac.uk/
[3] http://web.ornl.gov/sci/techresources/Human_Genome/index.shtml

Again, in Figure 3.11, we can see changes with almost equal insertions and deletion that lead to the indication of a renaming on the schemas elements and major restructuring of the schema. Those changes occur in mid 2001 and 2003, but probably in many more revisions which are difficult to detect due to the density of the figures and the limited space of this page. We also observe a massive alteration in the schemas elements from version 326 until version 329 that affects almost all the relations.

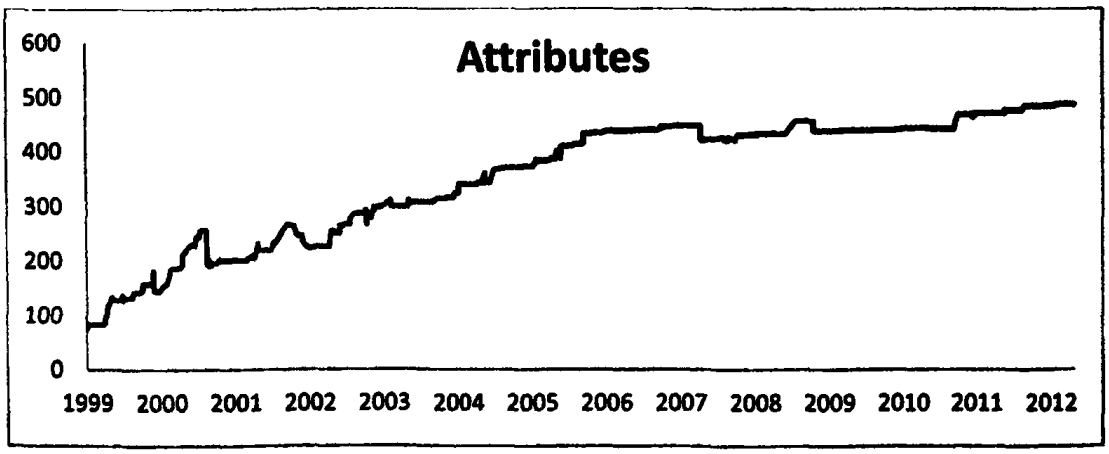
What is clearly visible about the number of the relations in Figure 3.12 is that it keeps growing with a diminishing rate as the schema gets more mature. Changes also diminish in both frequency and number. Comparing the number of relations with that of the attributes we see an almost linear growth but with many spikes and a generally a rough evolution.



a) Changes over Time for Ensembl

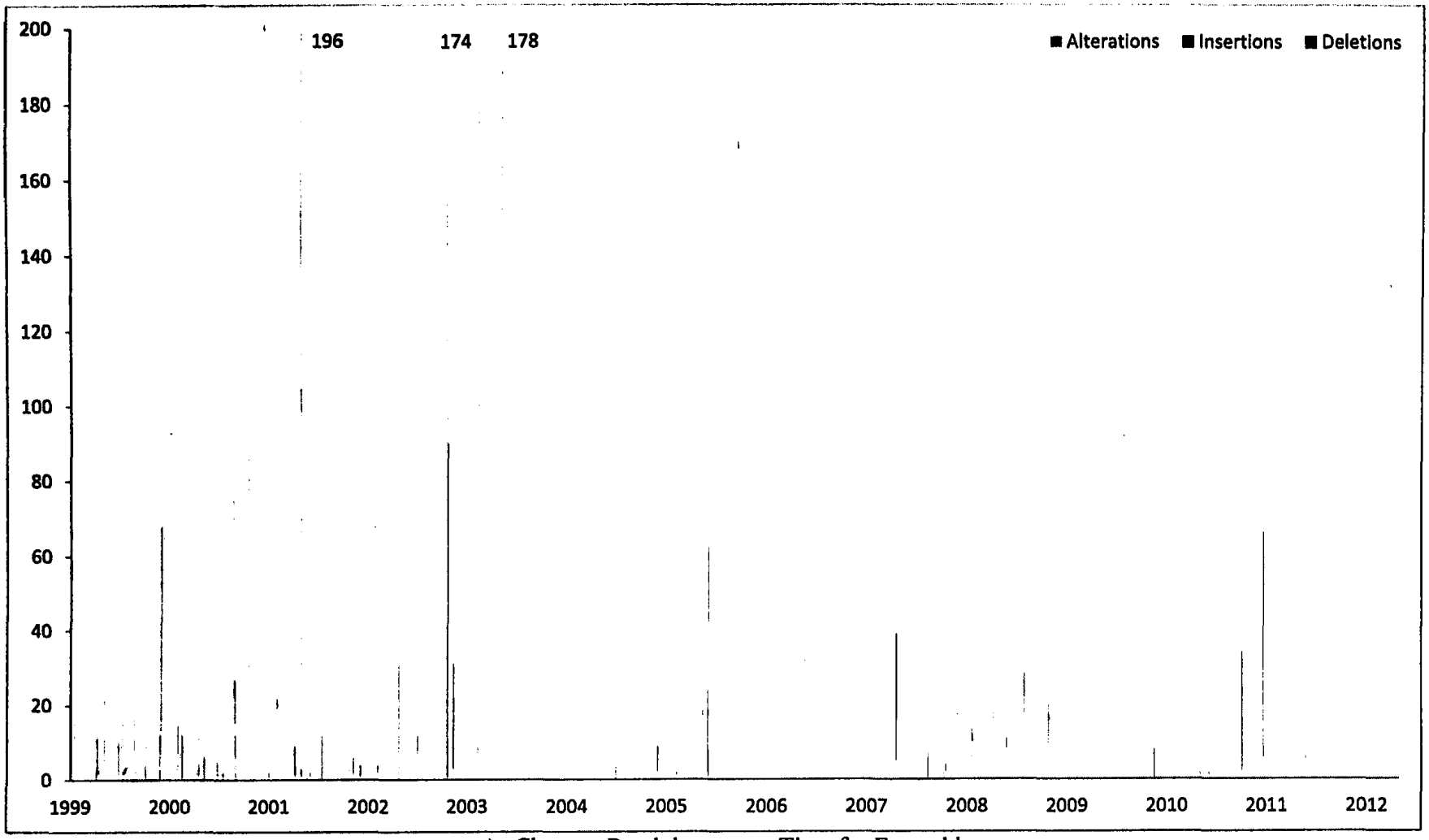


b) Number of Tables for Ensembl

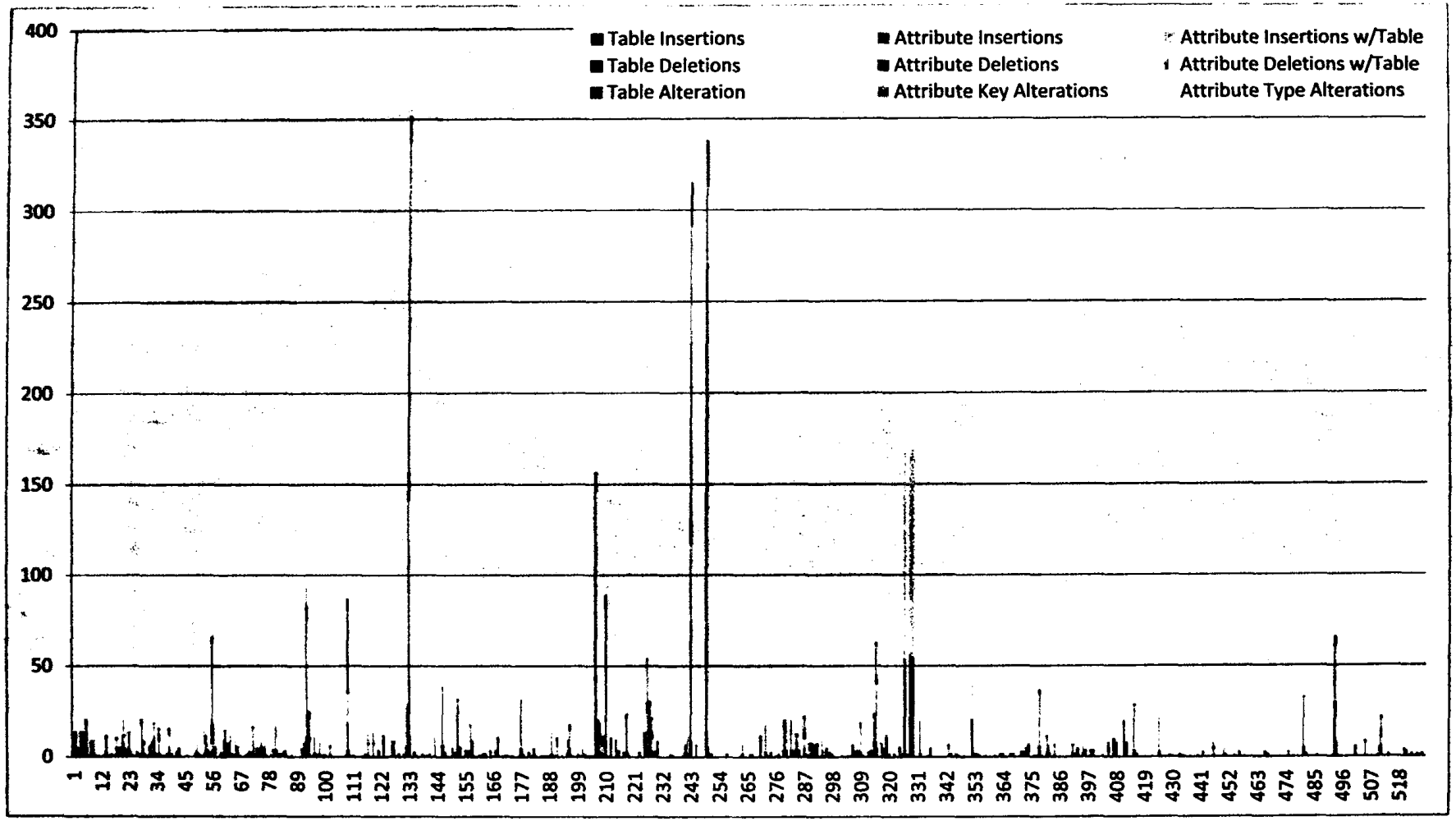


c) Number of Attributes for Ensembl

Figure 3.10 Events over time for Ensembl

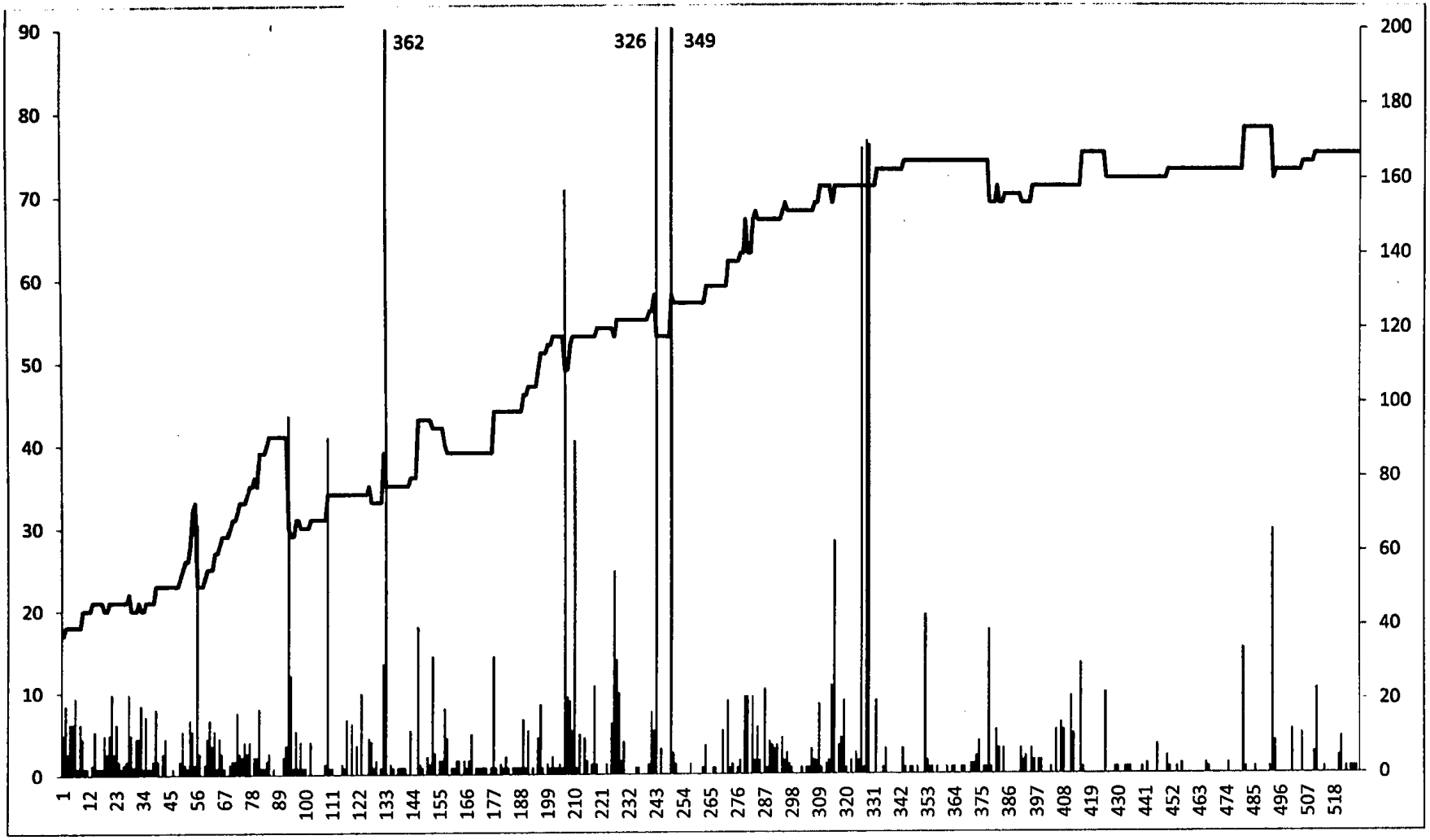


a) Changes Breakdown over Time for Ensembl

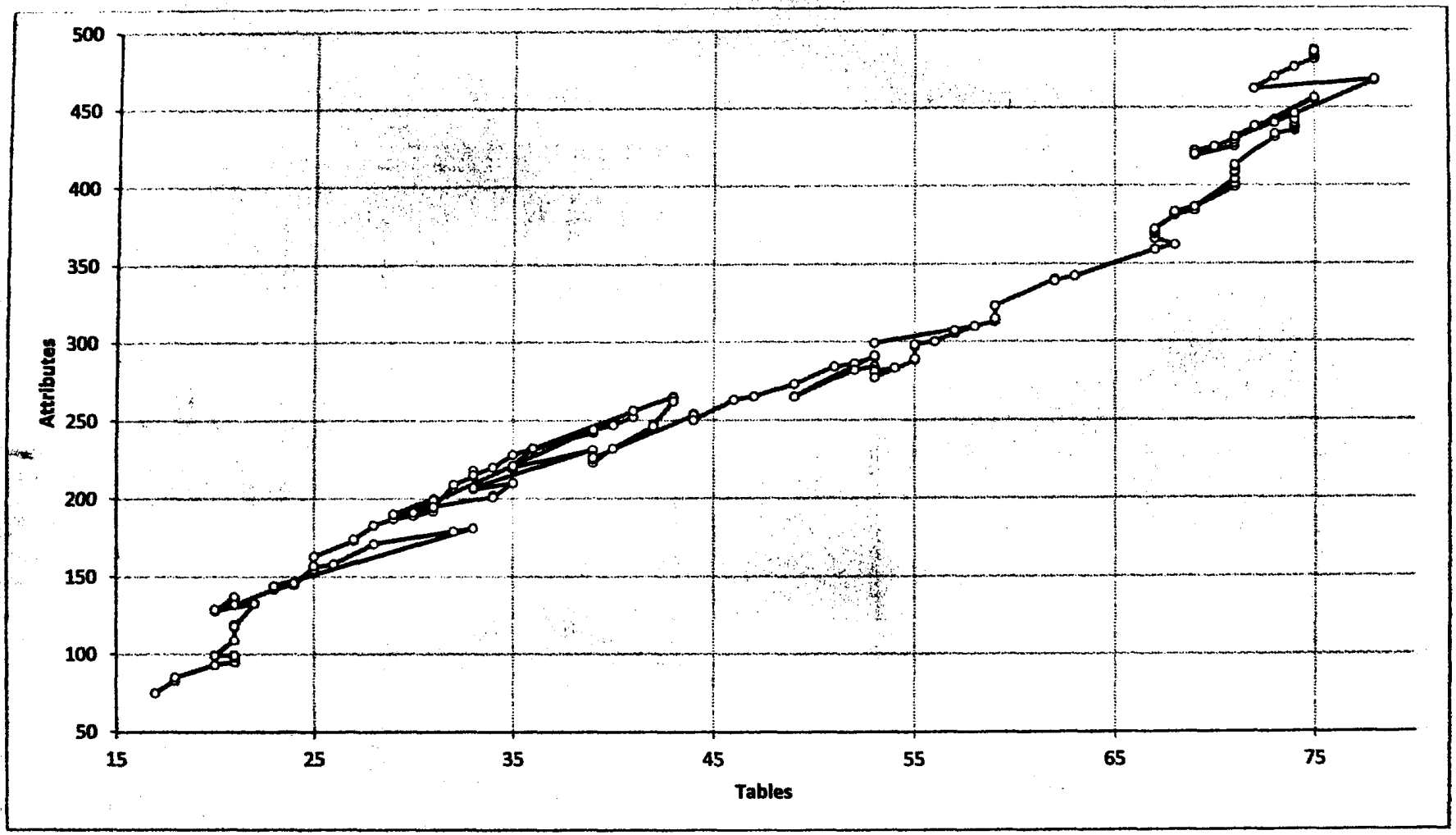


b) Changes Breakdown over Version ID for Ensembl

Figure 3.11 Analysis of Changes over Time for Ensembl



a) Changes compared with schema size for Ensembl



b) Timeline for Ensembl

Figure 3.12 Further insights for Ensembl

3.8. MediaWiki

MediaWiki[1] was first introduced in early 2002 by the Wikimedia Foundation[2] along with the famous Wikipedia[3], the free online collaboration encyclopedia and hosts Wikipedia's content since then. As an open source system (licensed under the GNU GPL) written in PHP, it was adopted by many companies and is used in thousands of websites. The software is optimized to efficiently handle large projects, which can have terabytes of content and hundreds of thousands of hits per second. It has also been deployed by some companies as an internal knowledge management system, and some educators have assigned students to use MediaWiki for collaborative group projects.

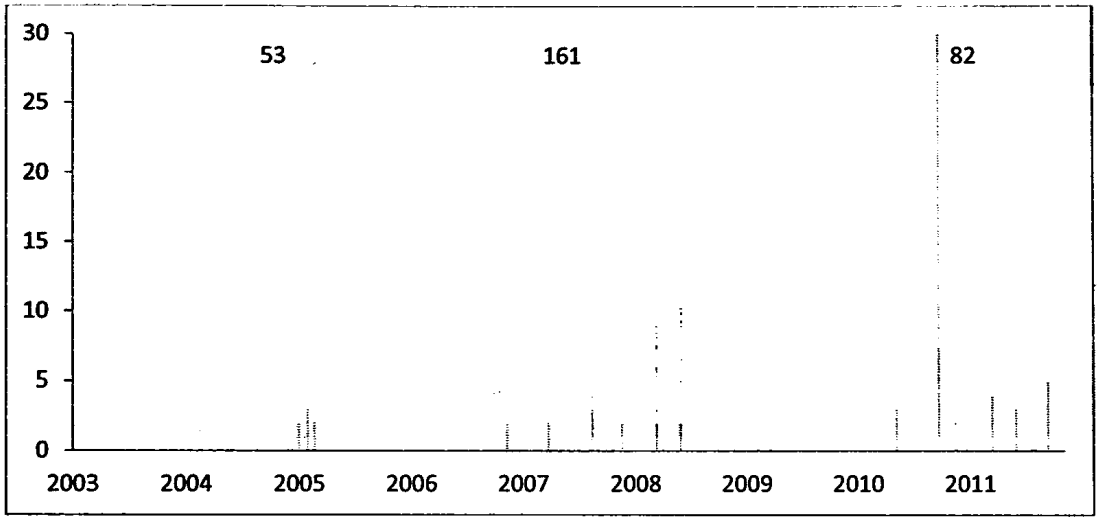
Once again, we used landscape pages for the change figures as we did for the Ensembl database. The numbers on the bars indicate the size of the change that was unable to fit in the figure

MediaWiki has a relatively smooth growth in size over time with many and frequent changes. The three highlights on Figure 3.13.a (noted with numbers on top) do not seem to affect the schema size and are most likely alteration or matched in number deletions and insertions. We also observe that every time the size of the relations drop is either after a previously inserted relation or followed by an insertion of a relation or relations. The only exception is at early 2005, when the project has very high activity and the relations number drops significantly, to recover after a year after. The attributes have a more steady growth with no major deletions.

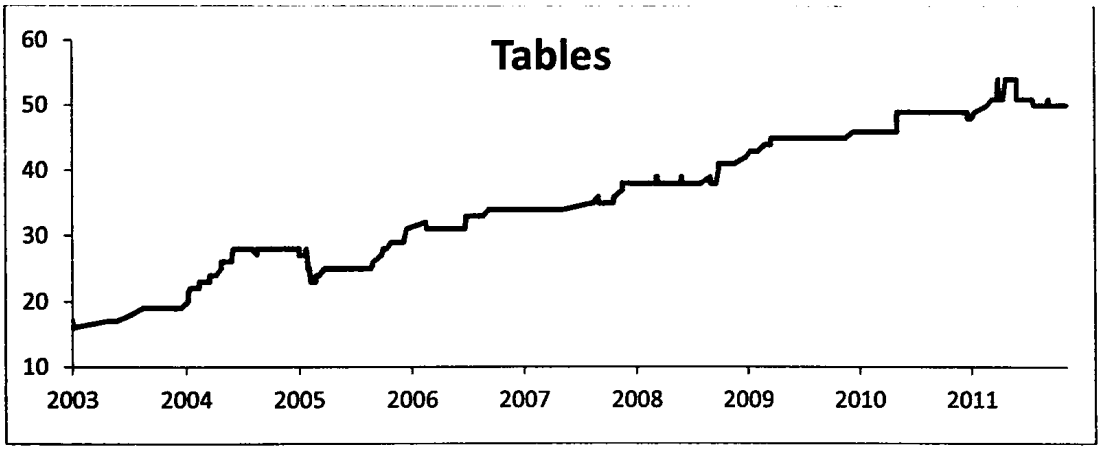
In Figure 3.14 we observe that the previously mentioned highlights are actually four. The first one is a major restructure/renaming of the schema. The following one, is a massive alteration to the schema elements that affects almost all the relations (32of 34 at that time). The last two, subsequent in order, are again alteration to the schema but with a smaller impact.

[1] https://www.mediawiki.org/wiki/MediaWiki
[2] https://wikimediafoundation.org/wiki/Home
[3] https://www.wikipedia.org/

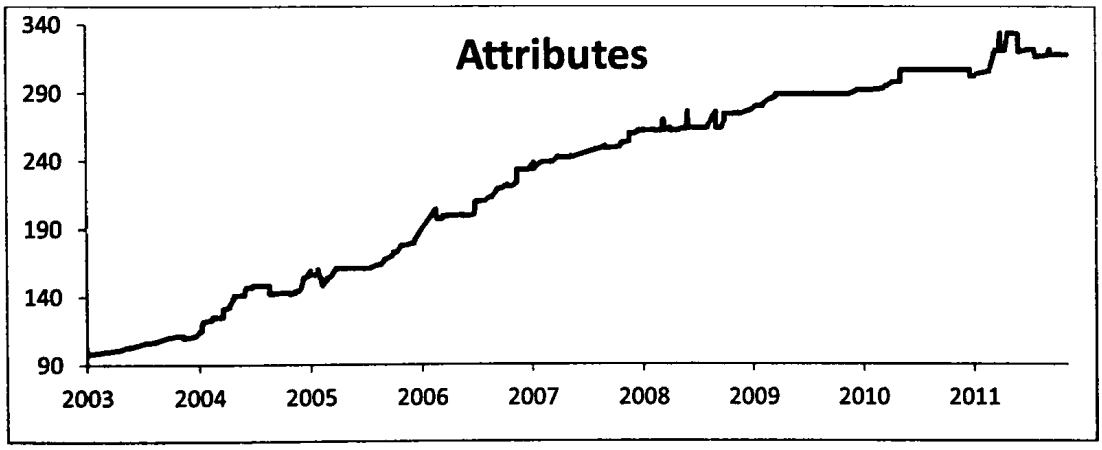
The size of the schema in terms of relations has a noteworthy drop closely after the great growth that follows the databases birth. In Figure 3.15 we observe small steps in the evolution of MediaWiki's database with no abrupt changes.



a) Changes Heartbeat over Time for MediaWiki

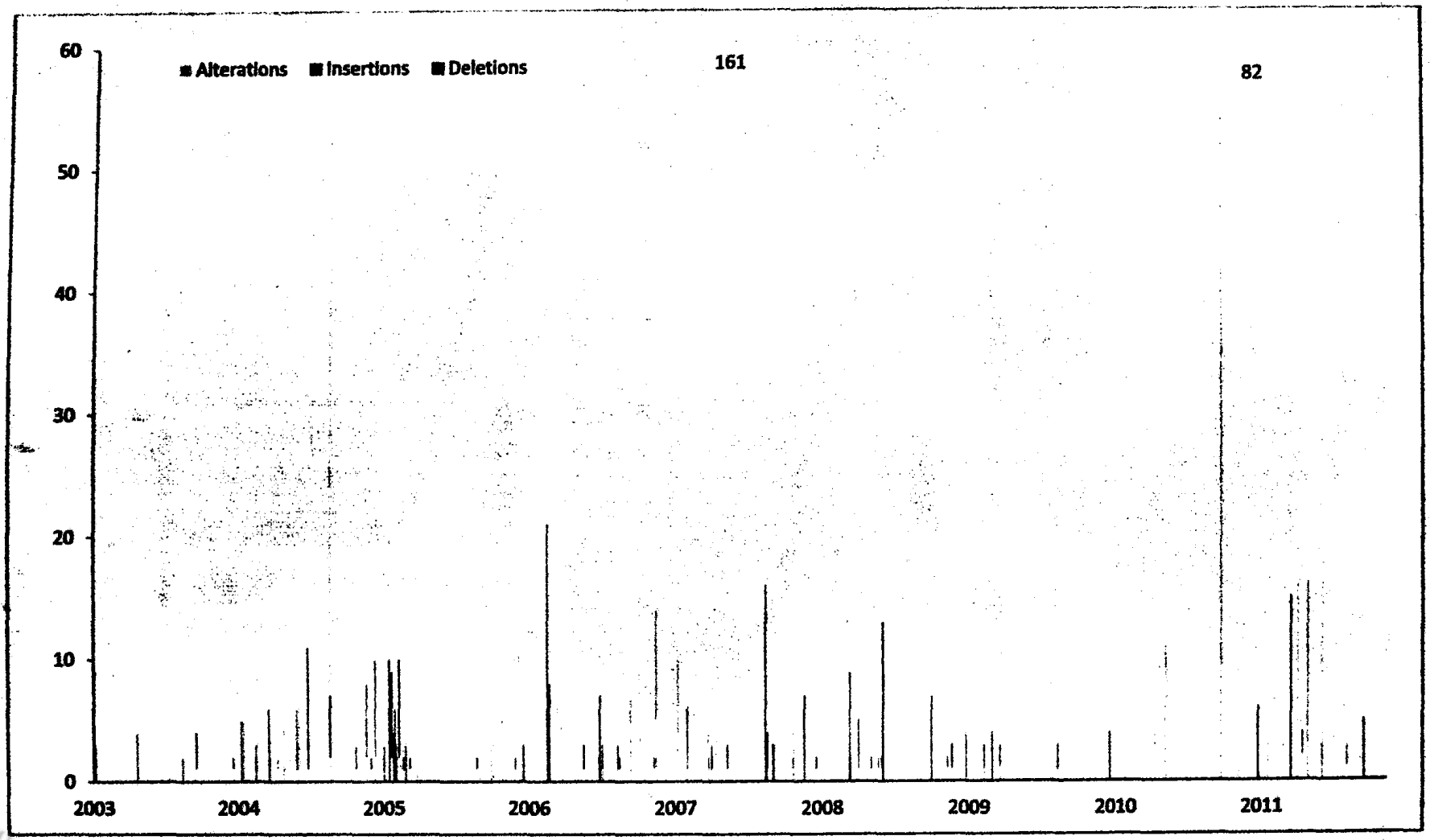


b) Number of Tables for MediaWiki

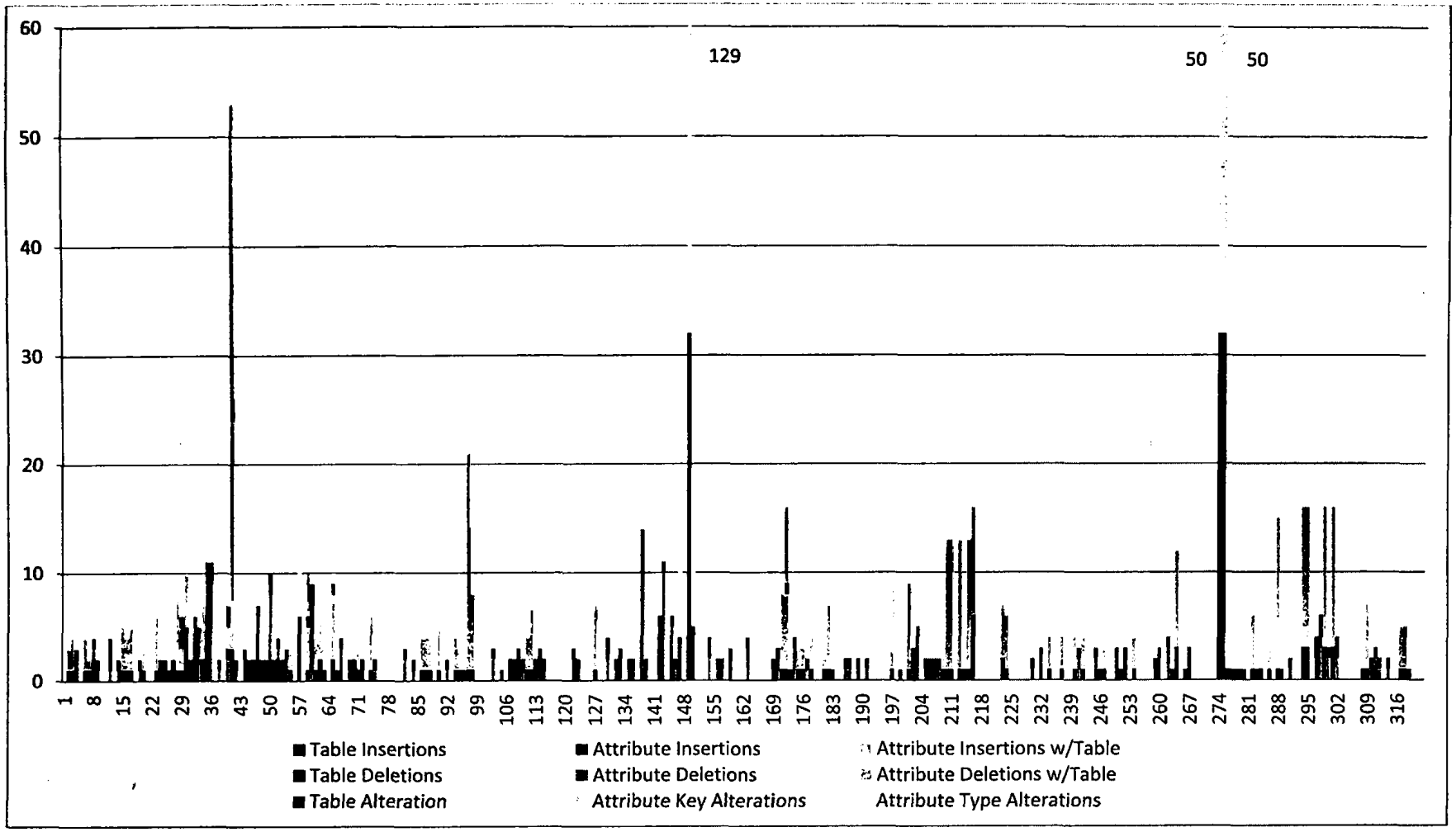


c) Number of Attributes for MediaWiki

Figure 3.13 Events over time for MediaWiki

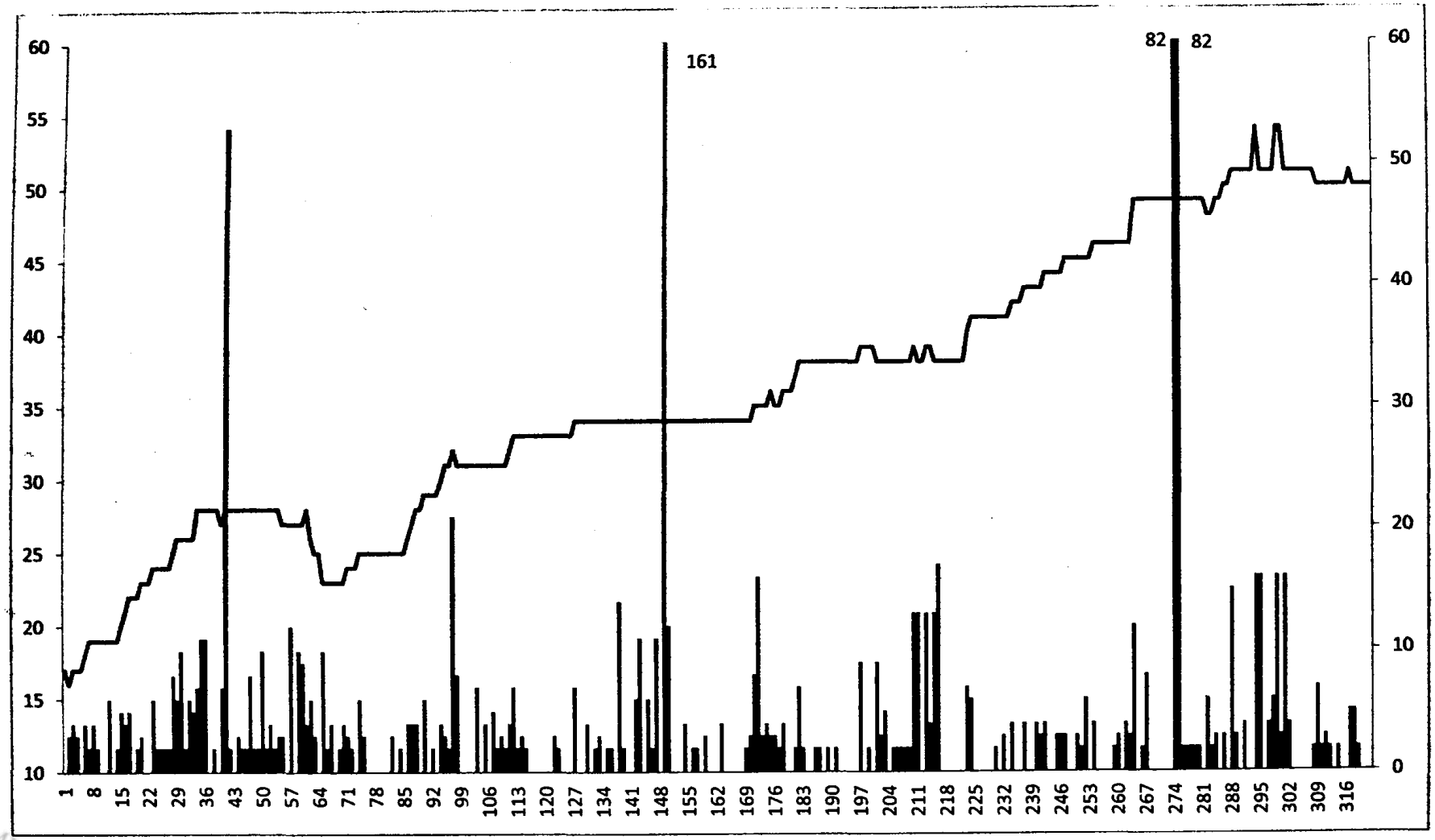


a) Changes Breakdown over Time for MediaWiki

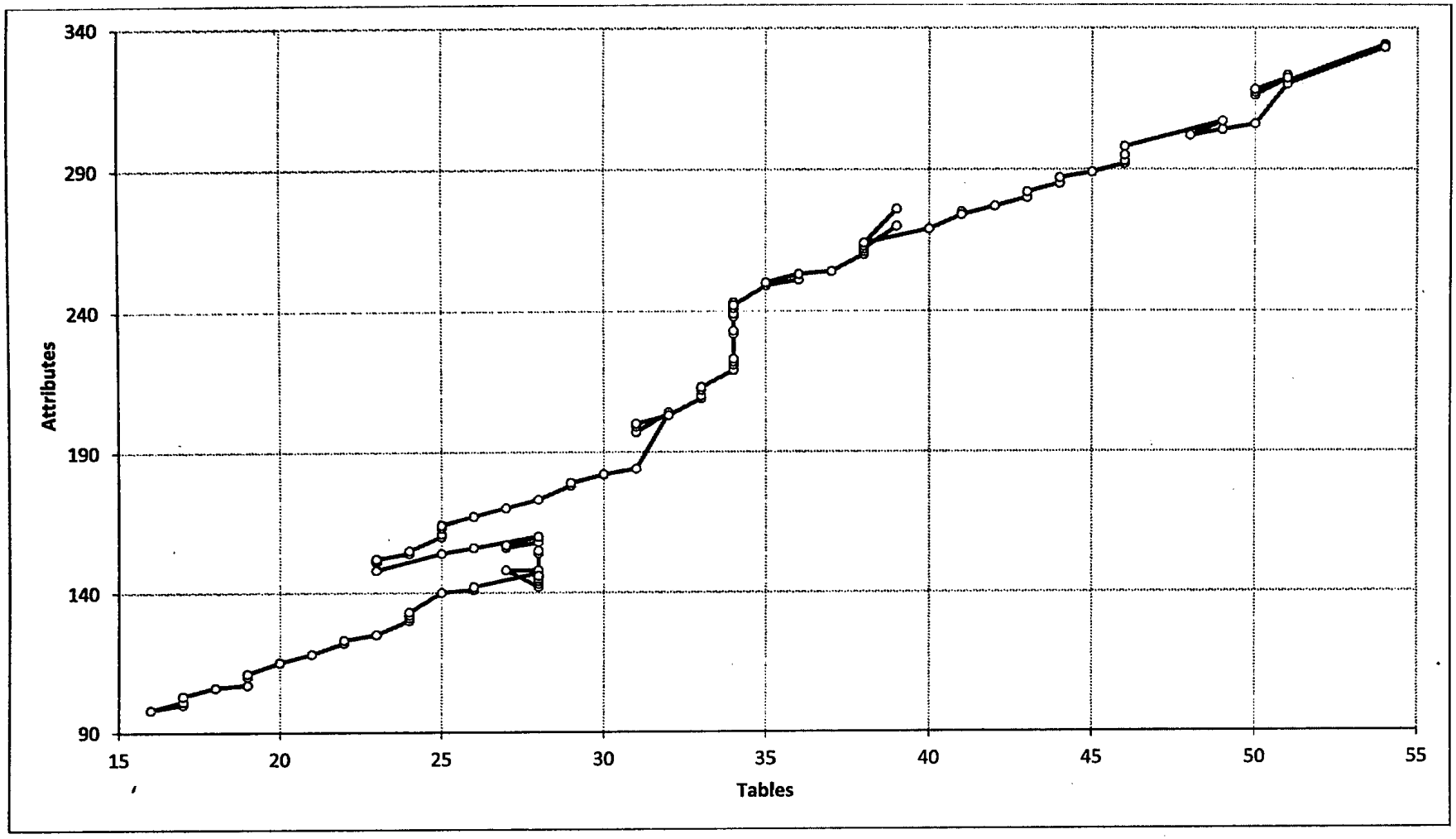


b) Changes Breakdown over Version ID for MediaWiki

Figure 3.14 Analysis of Changes over Time for MediaWiki



a) Changes compared with schema size for MediaWiki



b) Timeline for MediaWiki

Figure 3.15 Further insights for MediaWiki

3.9. OpenCart

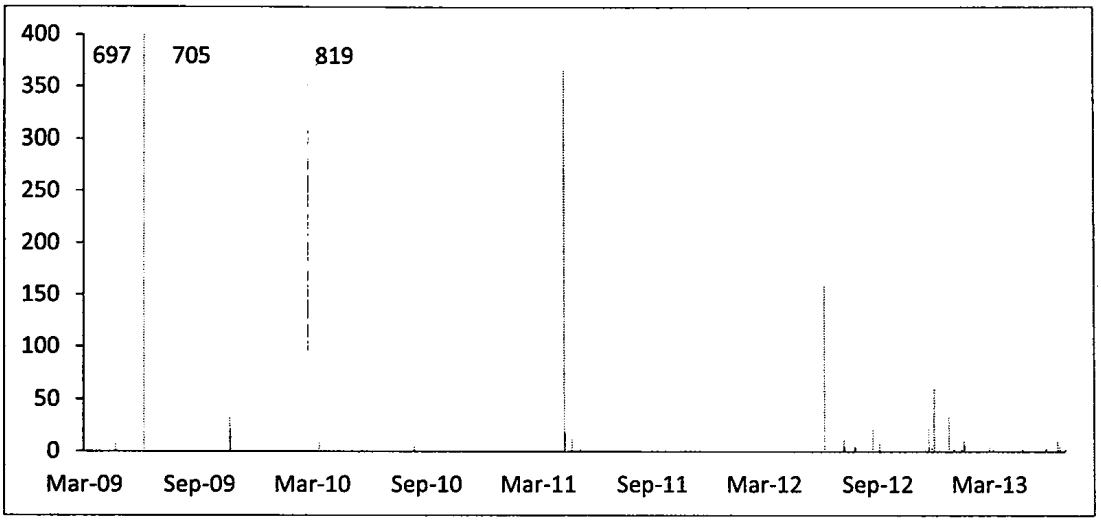
OpenCart[1] is an open source shopping cart system. It can be used on any web server with PHP and MySQL running on it. OpenCart is available as free software under the GNU General Public License. It is highly rated and used by many online sites.

From a first view of Figure 3.16 we see that the schema size has an almost strictly increasing growth, especially in terms of attributes. We notice three major increases is size; in early 2010, Q2 2011 and early 2012, generally at the start of each year. The three biggest changes (until March 2010) seem to affect the entire schema. A more detailed analysis is presented to the reader in the next set of figures.

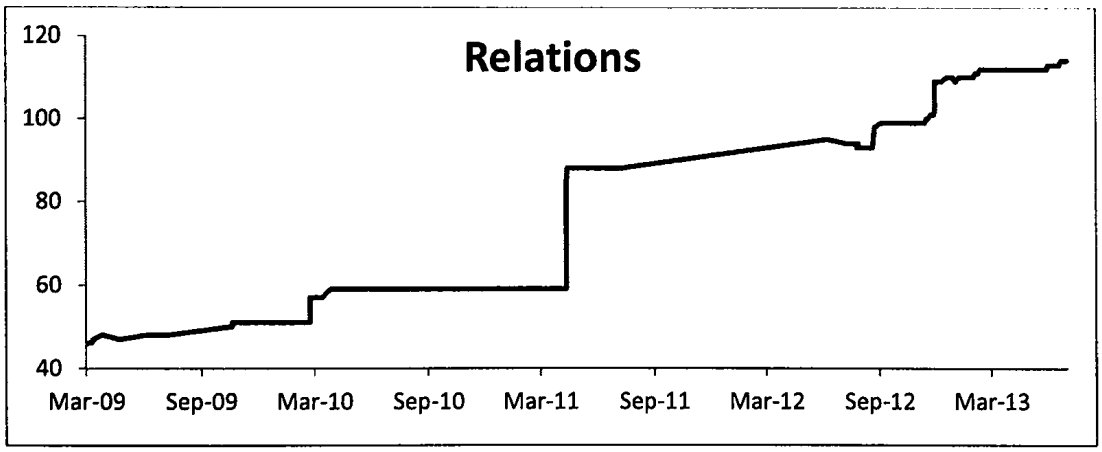
Figure 3.17 shows that the schema had mainly small changes. The tall bars on the figures have almost equal insertions and deletions thus leading to a conclusion of a major renaming of almost all the elements of the schema. The close to 350 insertions an deletions on revisions 17 and 18 as well as the more than 400 insertions an deletions on revision 23 match the size of the entire database schema. The majority of the rest of the changes are rather small compared to the size of the schema.

Again we confirm, in Figure 3.18, that the first two big changes had a very small impact to the schema size for the reason that they are renames to the schema elements. We also see more clearly the smooth almost strictly increasing of the schema size.

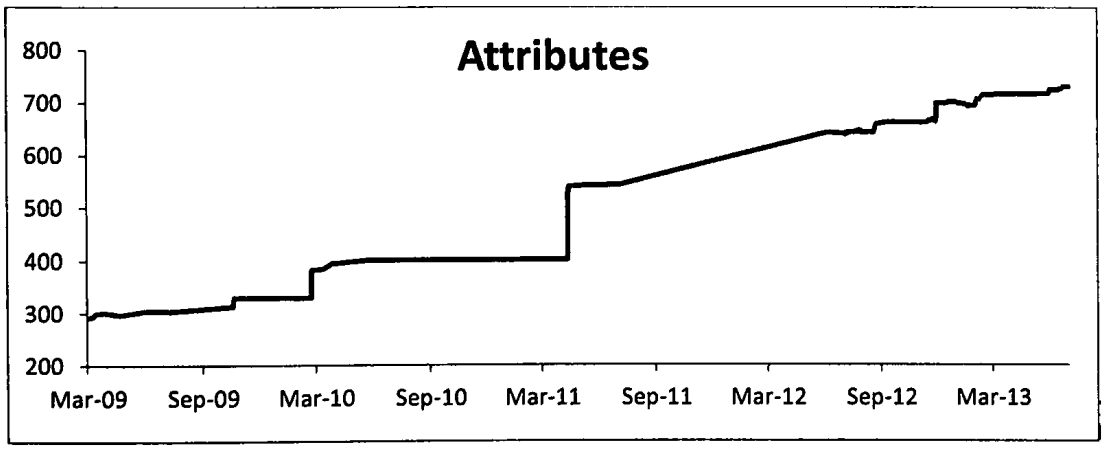
[1] http://www.opencart.com



a) Changes Heartbeat over Time for OpenCart

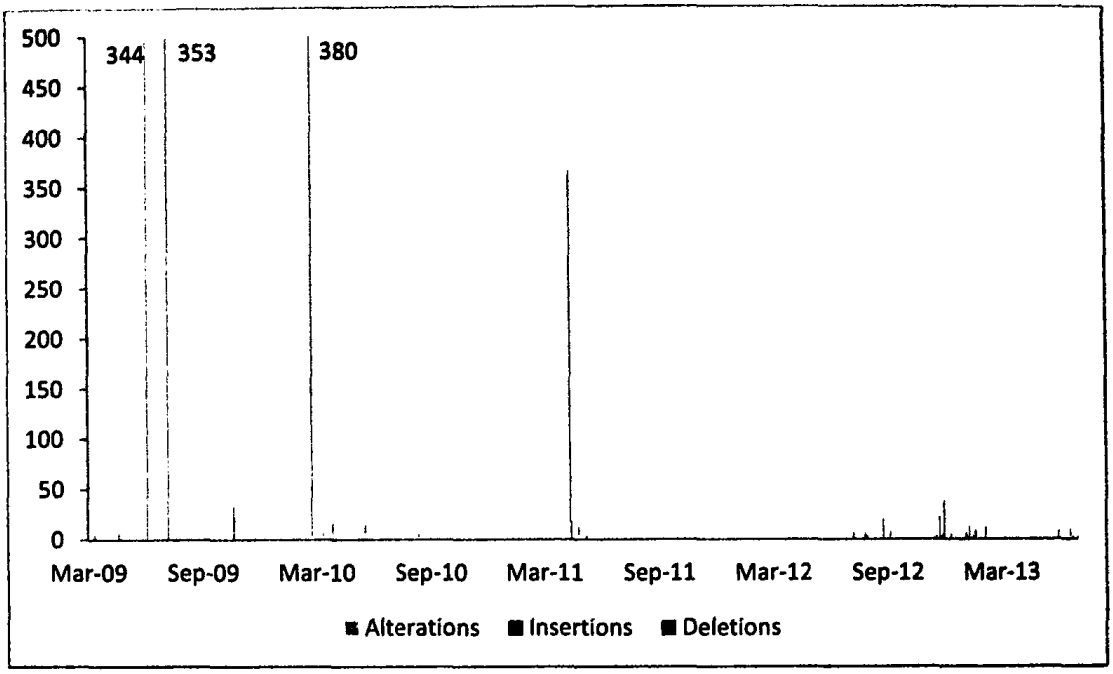


b) Number of Tables for OpenCart

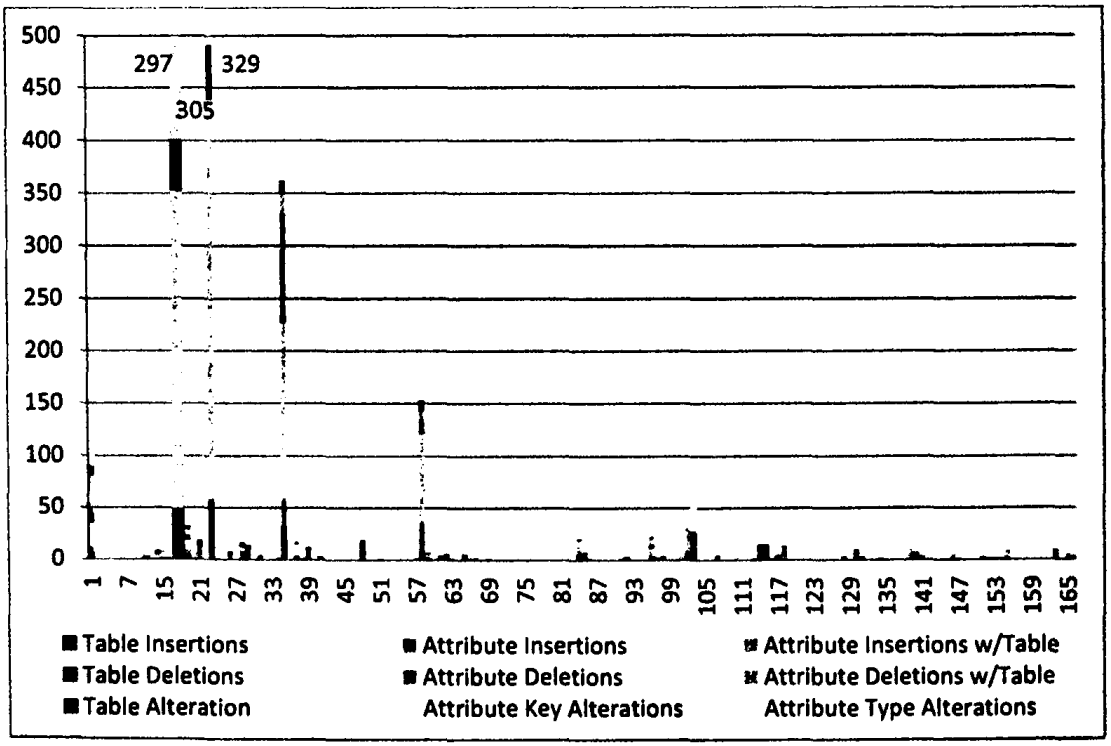


c) Number of Attributes for OpenCart

Figure 3.16 Events over time for OpenCart

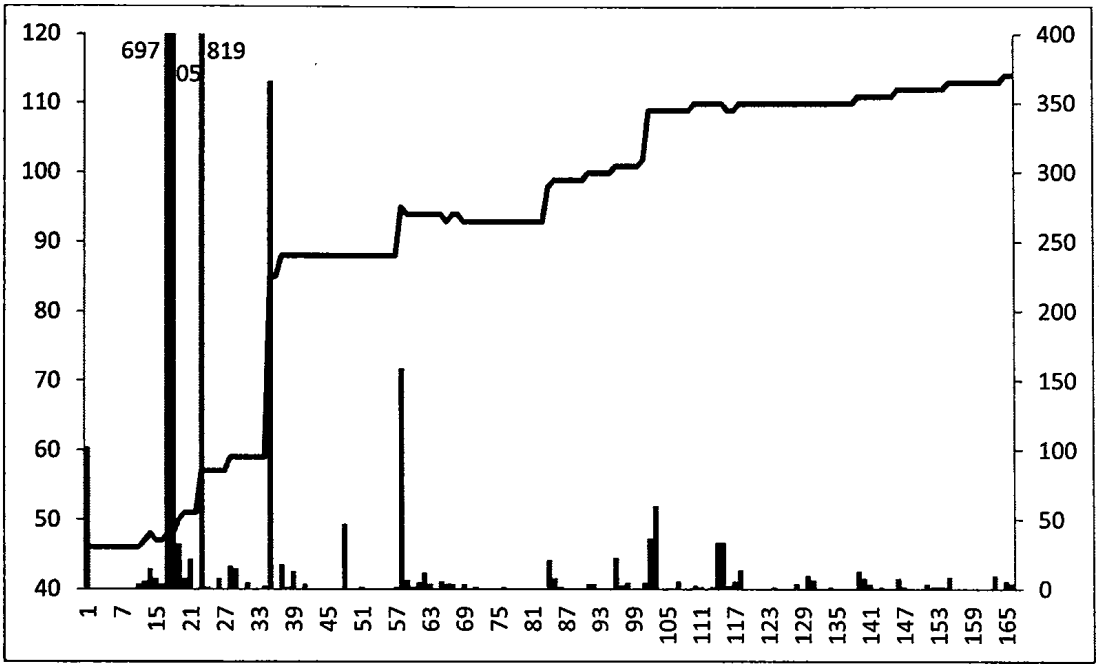


a) Changes Breakdown over Time for OpenCart

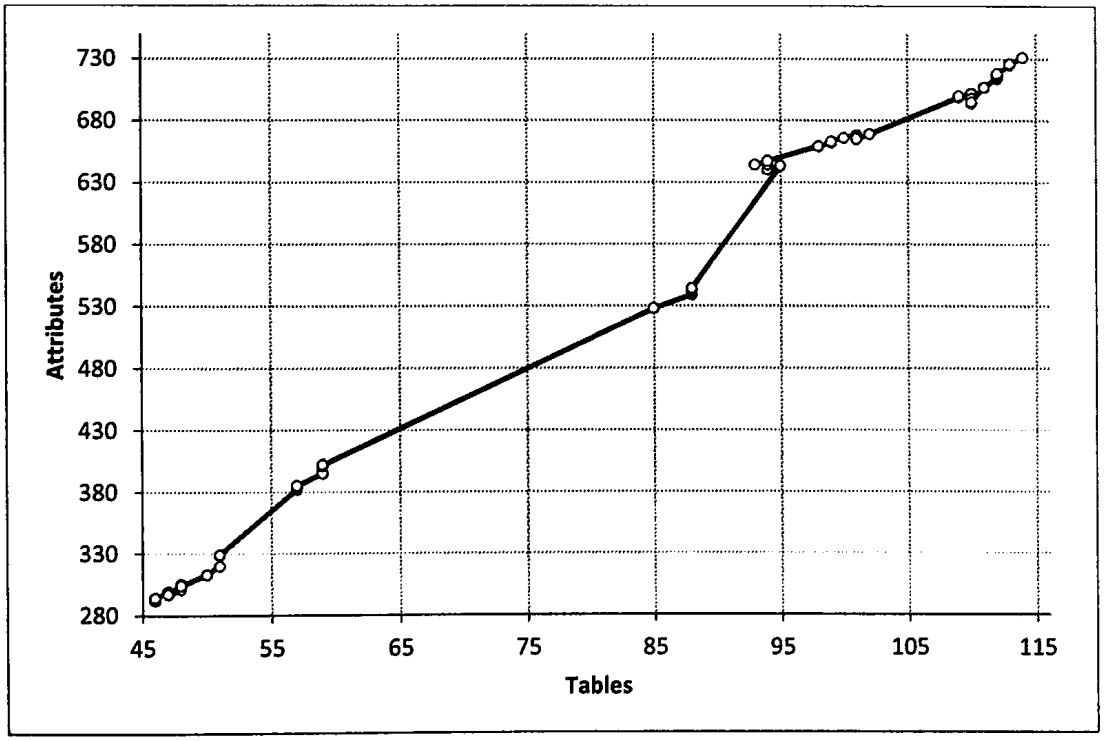


b) Changes Breakdown over Version ID for OpenCart

Figure 3.17 Analysis of Changes over Time for OpenCart



a) Changes compared with schema size for OpenCart



b) Timeline for OpenCart

Figure 3.18 Further insights for OpenCart

3.10. phpBB

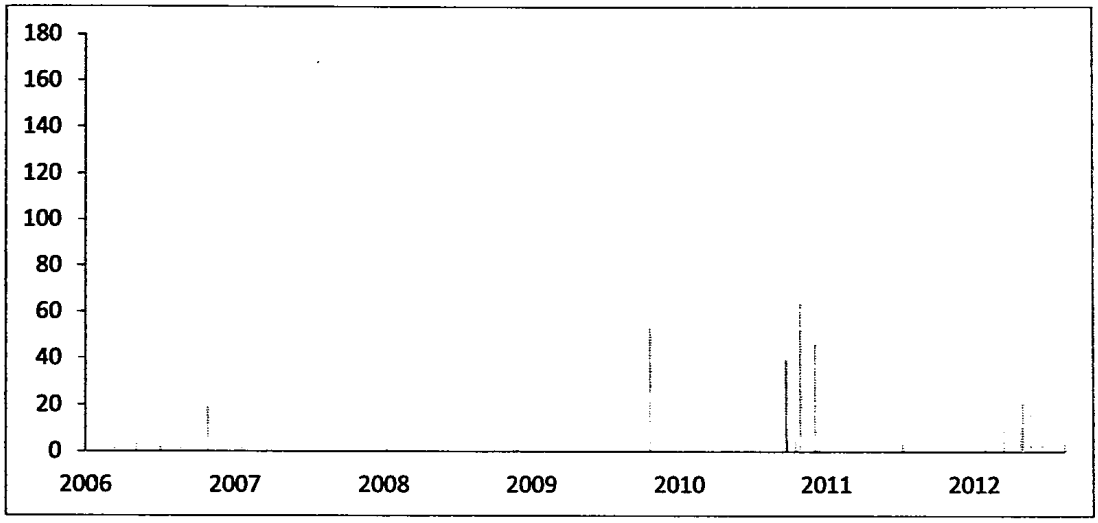
PhpBB[1] is an Internet forum package written in PHP. The name "phpBB" is an abbreviation of PHP Bulletin Board. It is available as free software under the GNU General Public License. PhpBB supports multiple database engines such as PostgreSQL, SQLite, MySQL, Oracle Database, and Microsoft SQL Server. It has a flat message structure (as opposed to threaded), hierarchical sub forums, topic split/merge/lock, user groups, multiple attachments per post, full-text search, plugins and various notification options (e-mail, Jabber instant messaging, ATOM feeds). It has a large community with an extensive number of user created modifications and styles and is one of the most commonly used open source forum software on the Internet. It was first released on December 16, 2000.

We consider the phpBB database an outlier to our study because of its peculiar behavior we can examine in Figure 3.19, Figure 3.20 and Figure 3.21. It's the only case from those we researched that the schema had an overall drop on the number of its elements with a quite intensive period the last two years. The number of the attributes suffers a big drop in mid 2006 that was never recovered in its six years of life accompanied with a growth in relations.

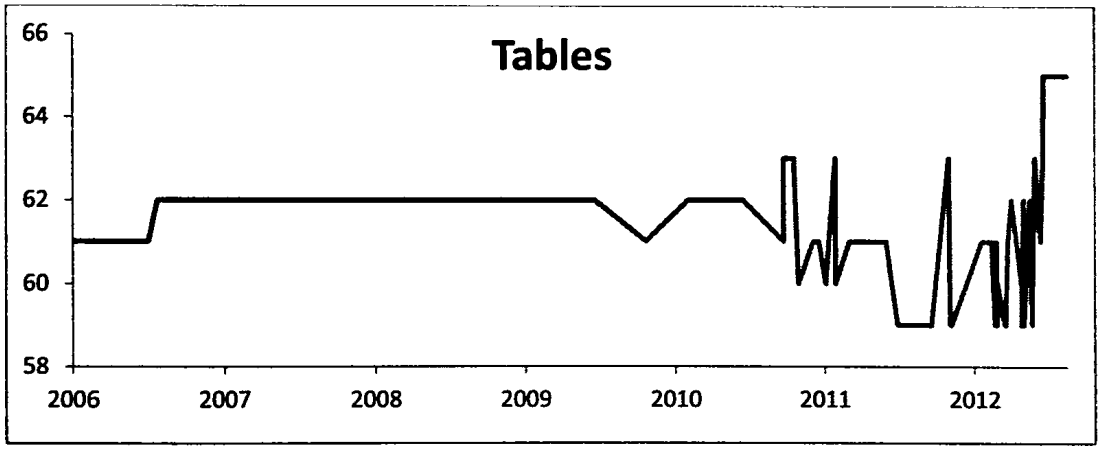
The first half of its life (Figure 3.20), the schema had, apart from one major decrement in attributes, many alterations without affecting the schema size. From late 2009 (revision 65) the project had an increasing activity with subsequent insertions and deletion that kept the schema size almost stable.

In Figure 3.21.b the first version of this database schema, in contrast with what we observe in the timeline figures of the rest of the databases, has its first version not at the left bottom side of the figure but at the top (approximately 610 attributes and 61 relations). Again, we can see the rough change in the schema size that keeps going back and forth without major changes especially in terms of relations. The changes we observe are most of the time insertions and deletions of a single relation, but greater in number for attributes.

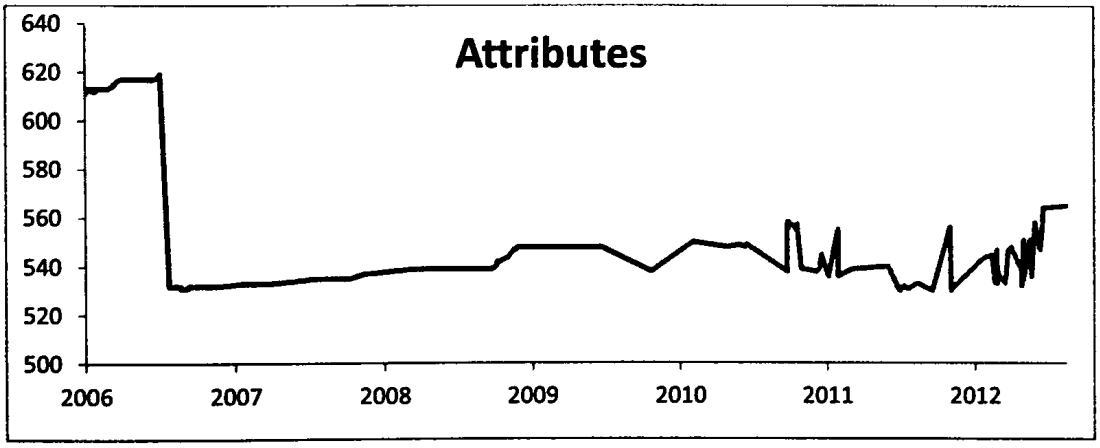
[1] https://www.phpbb.com/



a) Changes Heartbeat over Time for phpBB

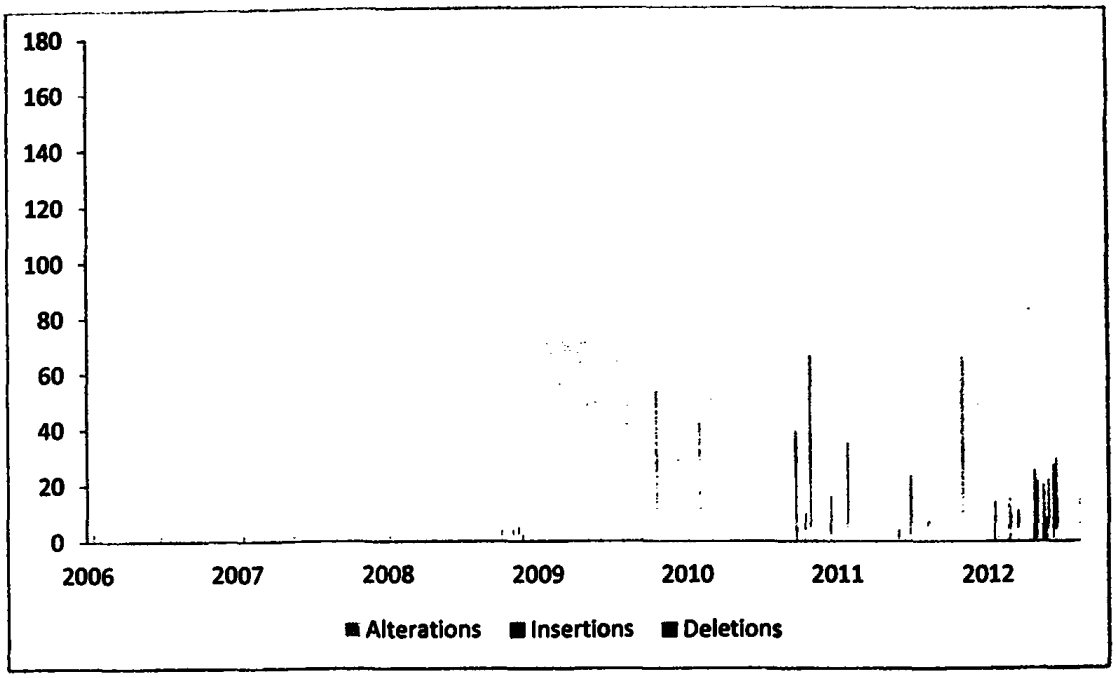


b) Number of Tables for phpBB

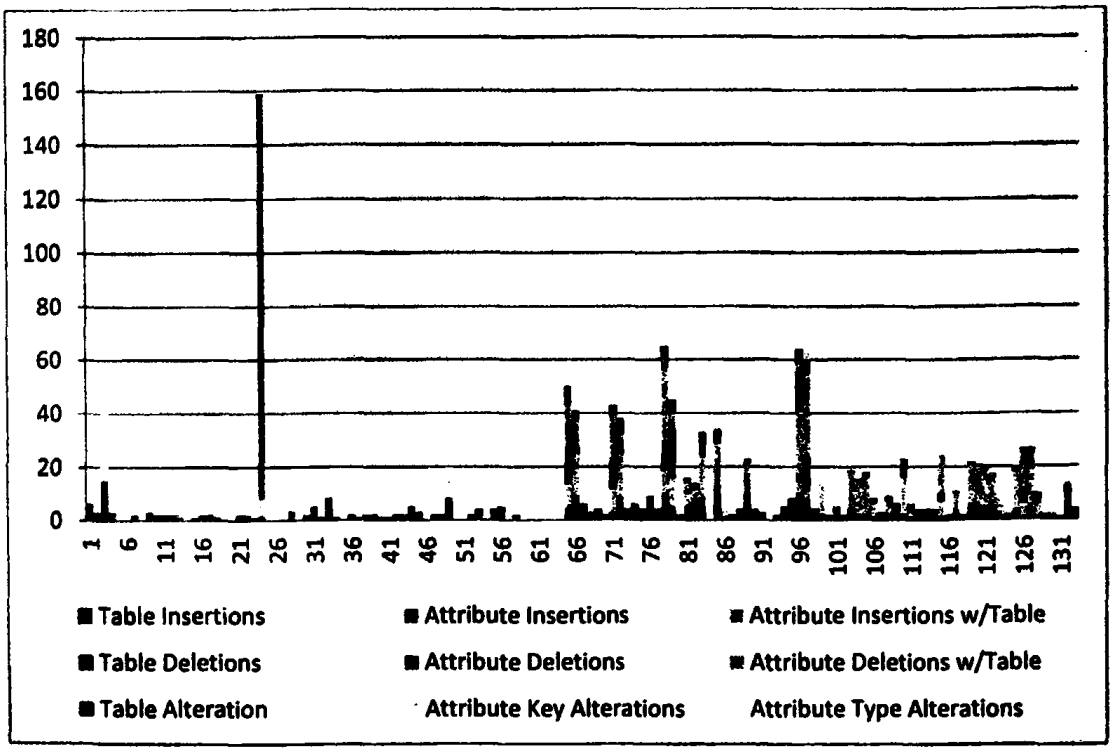


c) Number of Attributes for phpBB

Figure 3.19 Events over time for phpBB

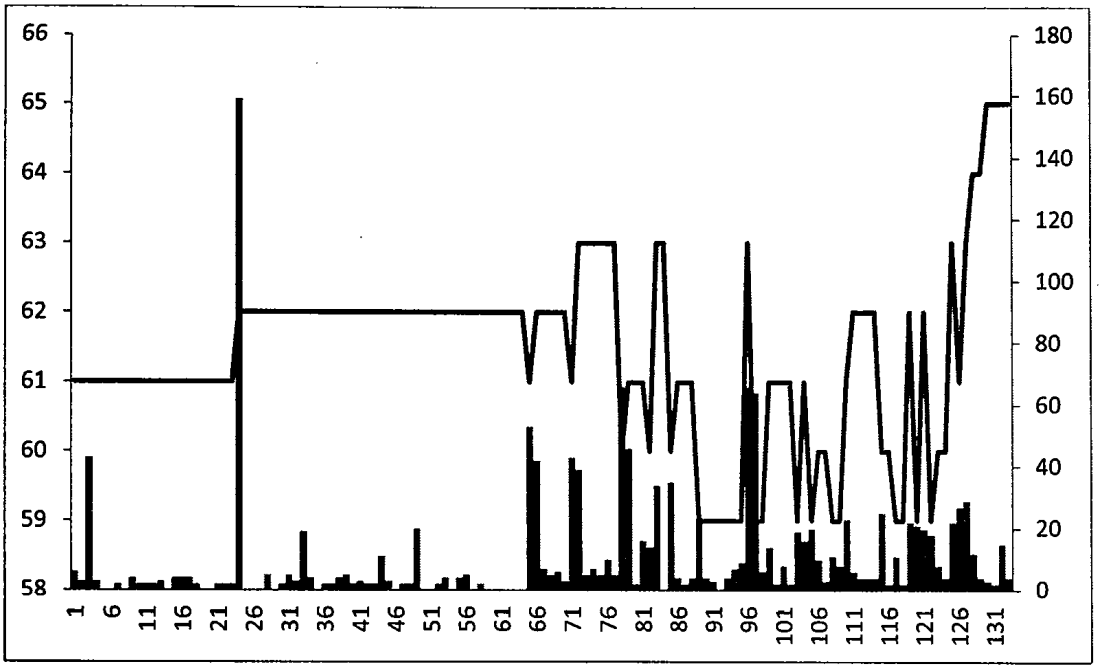


a) Changes Breakdown over Time for phpBB

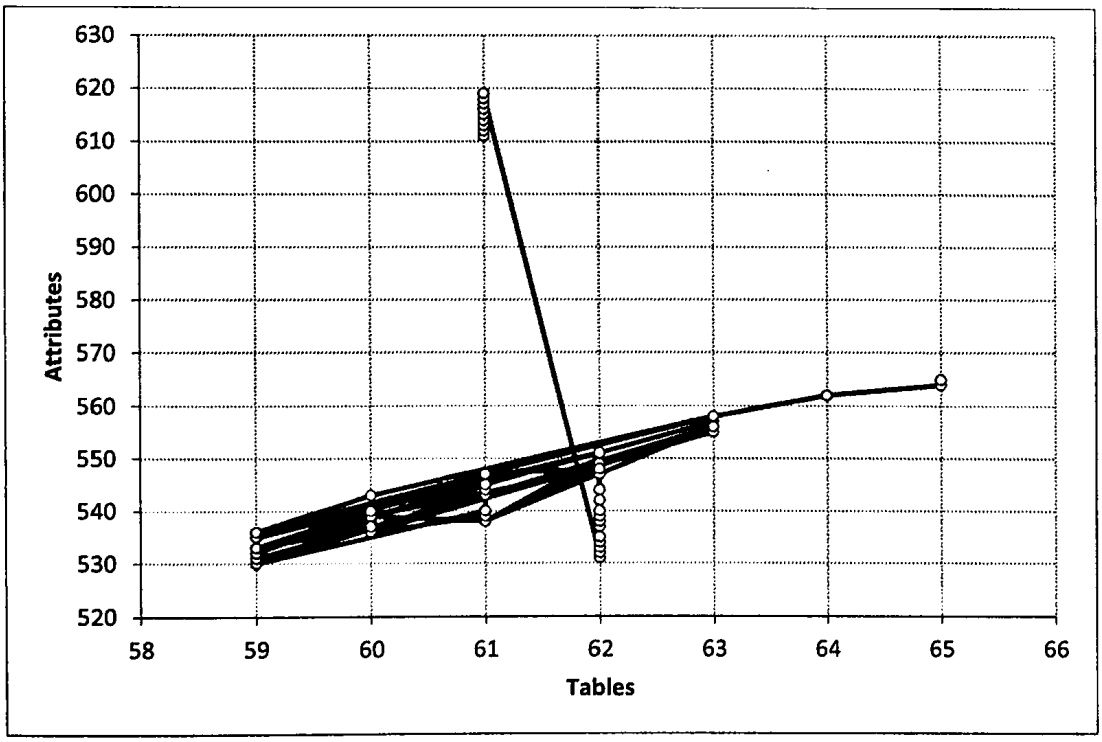


b) Changes Breakdown over Version ID for phpBB

Figure 3.20 Analysis of Changes over Time for phpBB



a) Changes compared with schema size for phpBB



b) Timeline for phpBB

Figure 3.21 Further insights for phpBB

3.11. TYPO3

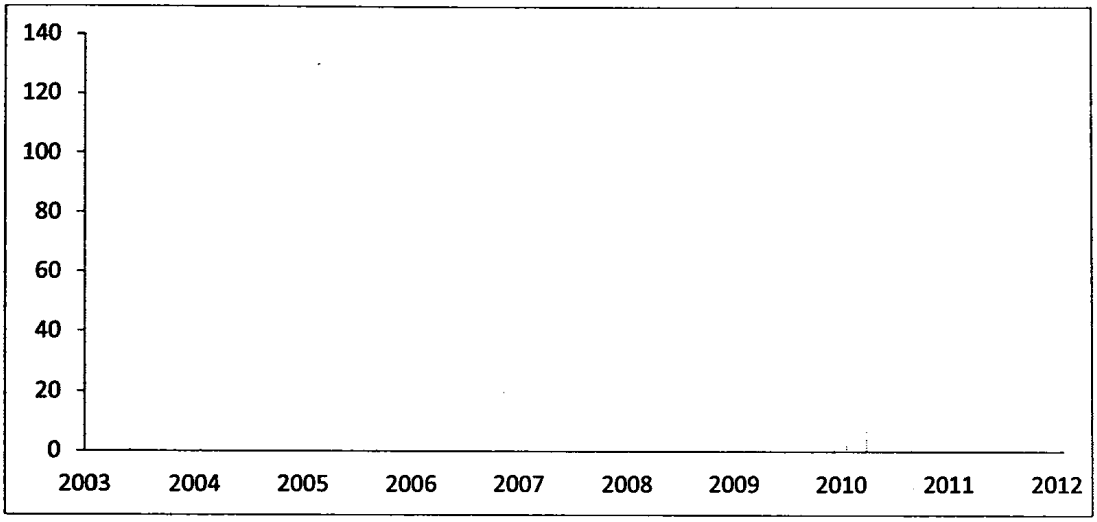
TYPO3[1] is a free and open source web content management framework based on PHP. It is released under the GNU General Public License and can run on several web servers, such as Apache or IIS, on top of many operating systems, among them Linux, Microsoft Windows, FreeBSD, Mac OS X and OS/2. TYPO3 is, along with Drupal, Joomla and Wordpress, among the most popular content management systems worldwide, however it is more widespread in Europe than in other regions. TYPO3 was initially authored by Kasper Skårhøj in 1997. Having a unique, to the datasets that we have seen so far, strict growth in both relations and attributes, TYPO3 had a three year respite with no change in relations size and some minor insertions in attributes. The generally continuous growing number of relations was interrupted in 2010 with a considerable drop that almost halved their number just to rise again along with a great increase in the number of attributes.

TYPO3's activity, as observed in Figure 3.22, is quite low compared to our previous datasets. The scarce changes that we see in Figure 3.22.a are, most likely, additions until late 2009 as we fail to observe drops in the number of relations and attributes (Figure 3.22.b, Figure 3.22.c). After 2009 we observe an increase in the activity of the project with the number of relation having a significant drop, something that does not happen in the number of the attributes. After the first quarter of 2011, TYPO3's database recovers the relations that were previously lost along with a tremendous increase in attributes that until the end are almost doubled.

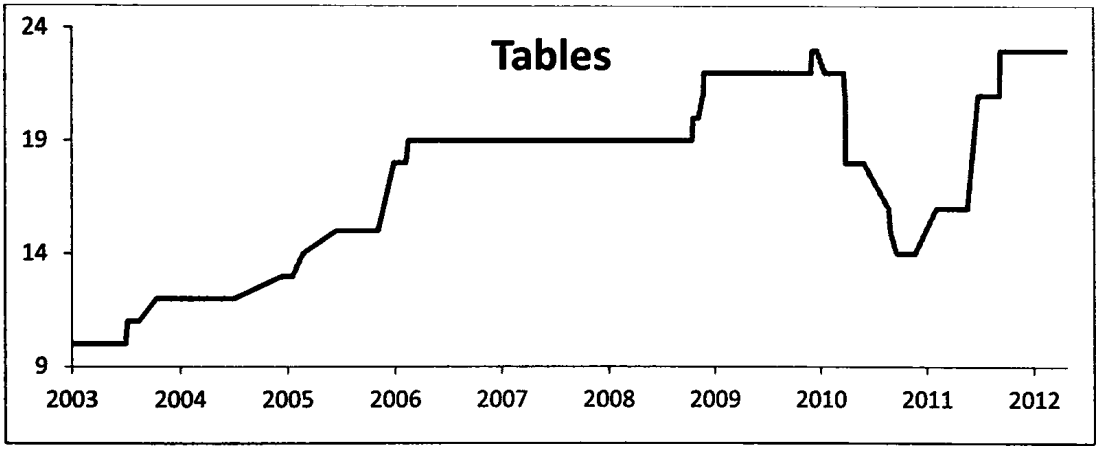
In Figure 3.23.a we can clearly see the dominance of blue (additions). Deletions appear only after 2008 (revision 21).

In Figure 3.24 we can again see the great drop in the number of tables.

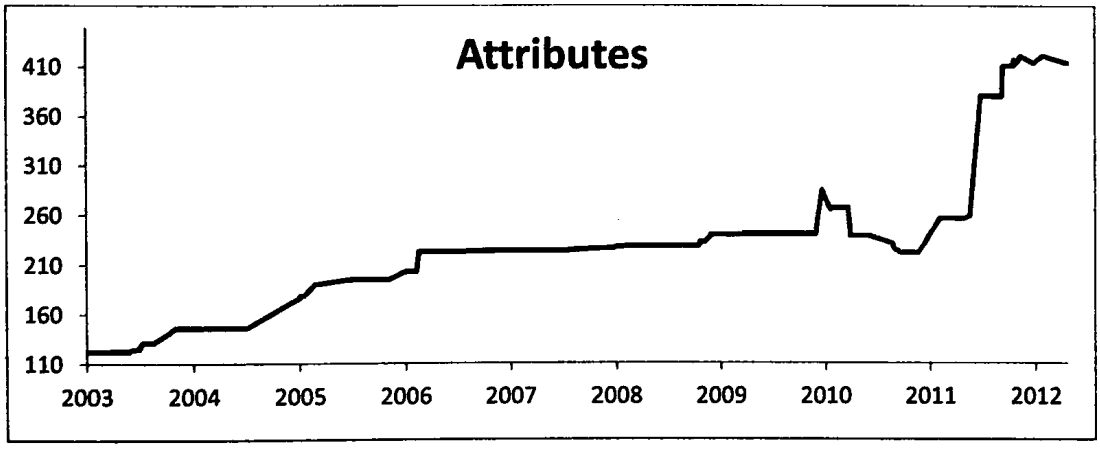
[1] http://typo3.org/



a) Changes Heartbeat over Time for TYPO3

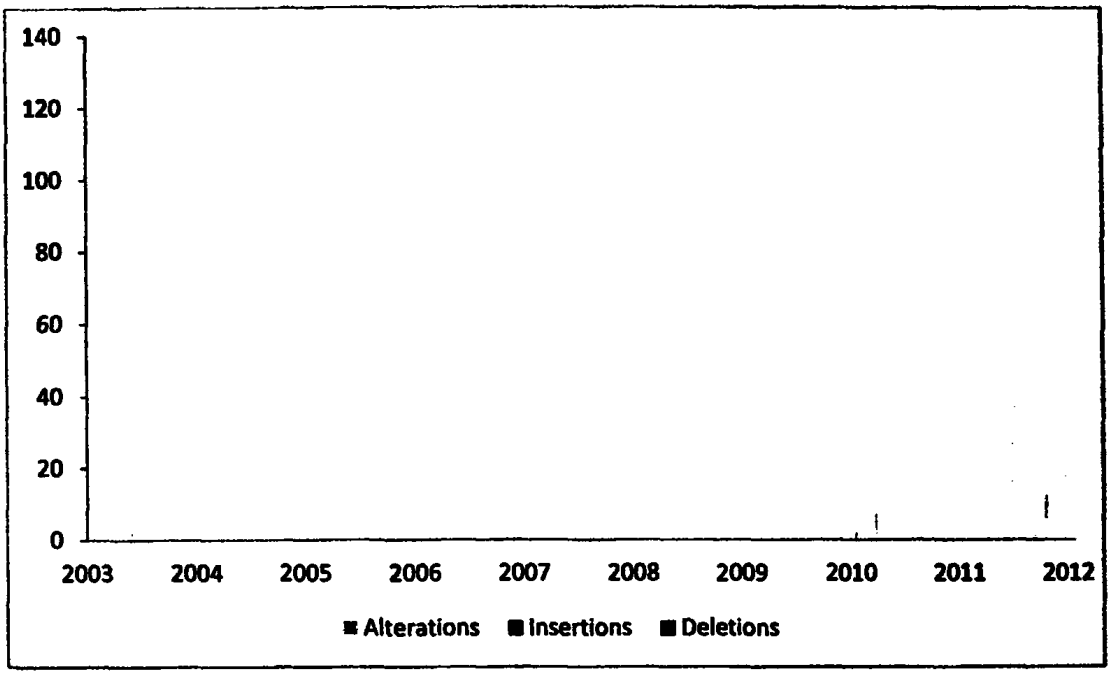


b) Number of Tables for TYPO3

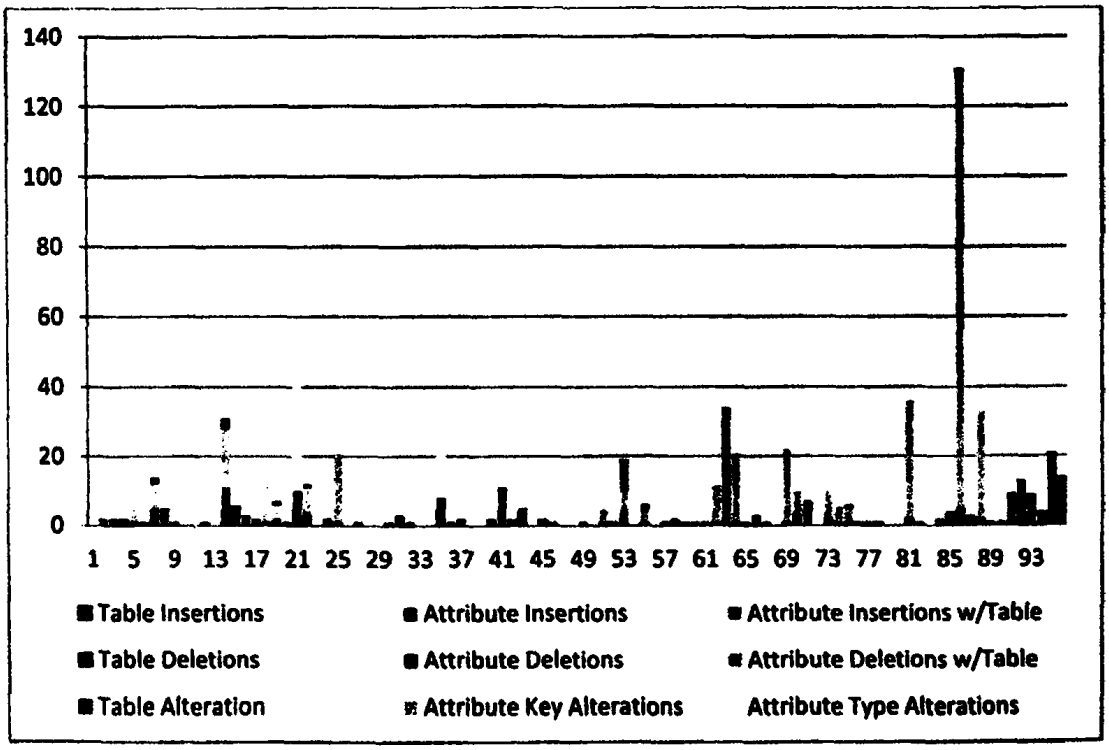


c) No. of Attributes for TYPO3

Figure 3.22 Events over time for TYPO3

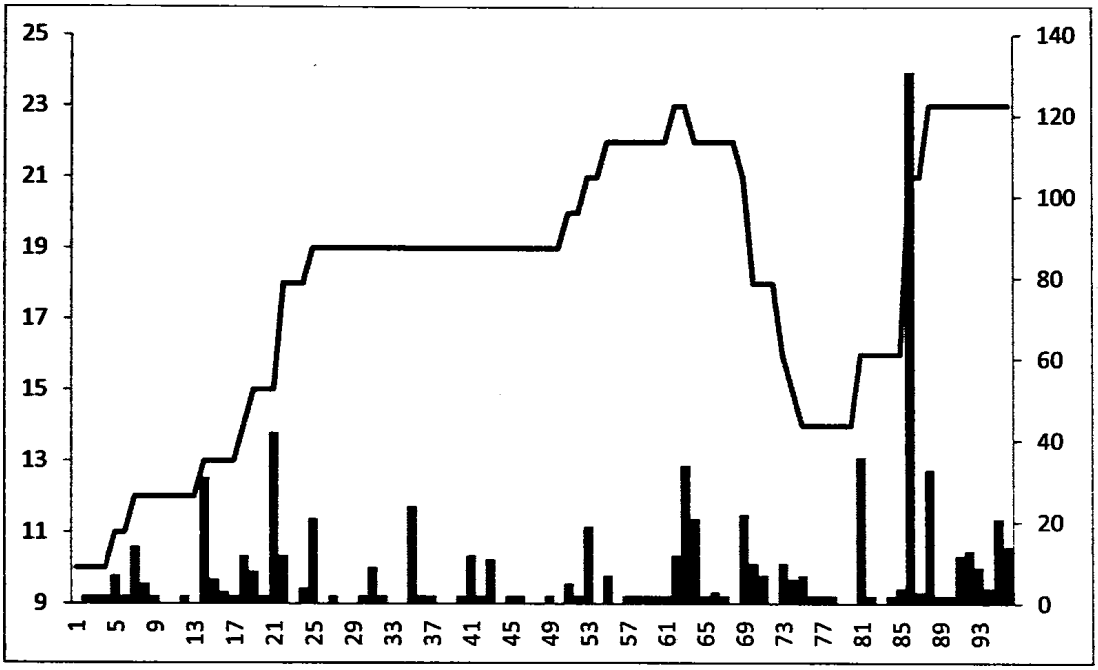


a) Changes Breakdown over Time for TYPO3

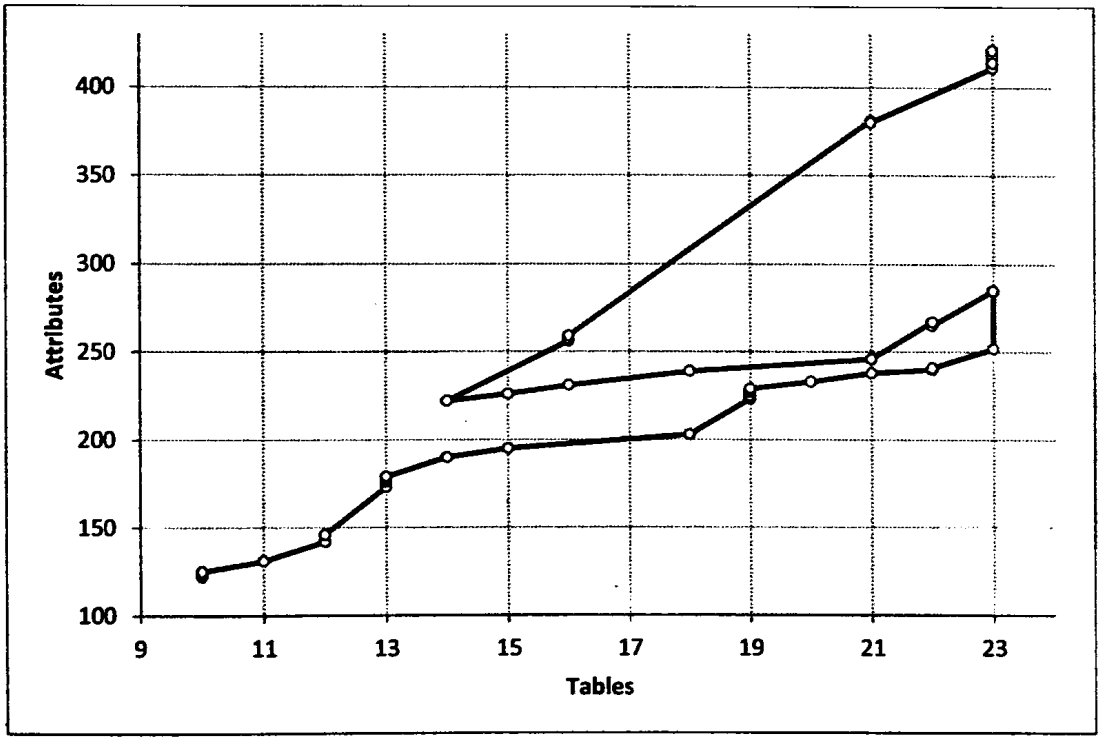


b) Changes Breakdown over Version ID for TYPO3

Figure 3.23 Analysis of Changes over Time for TYPO3

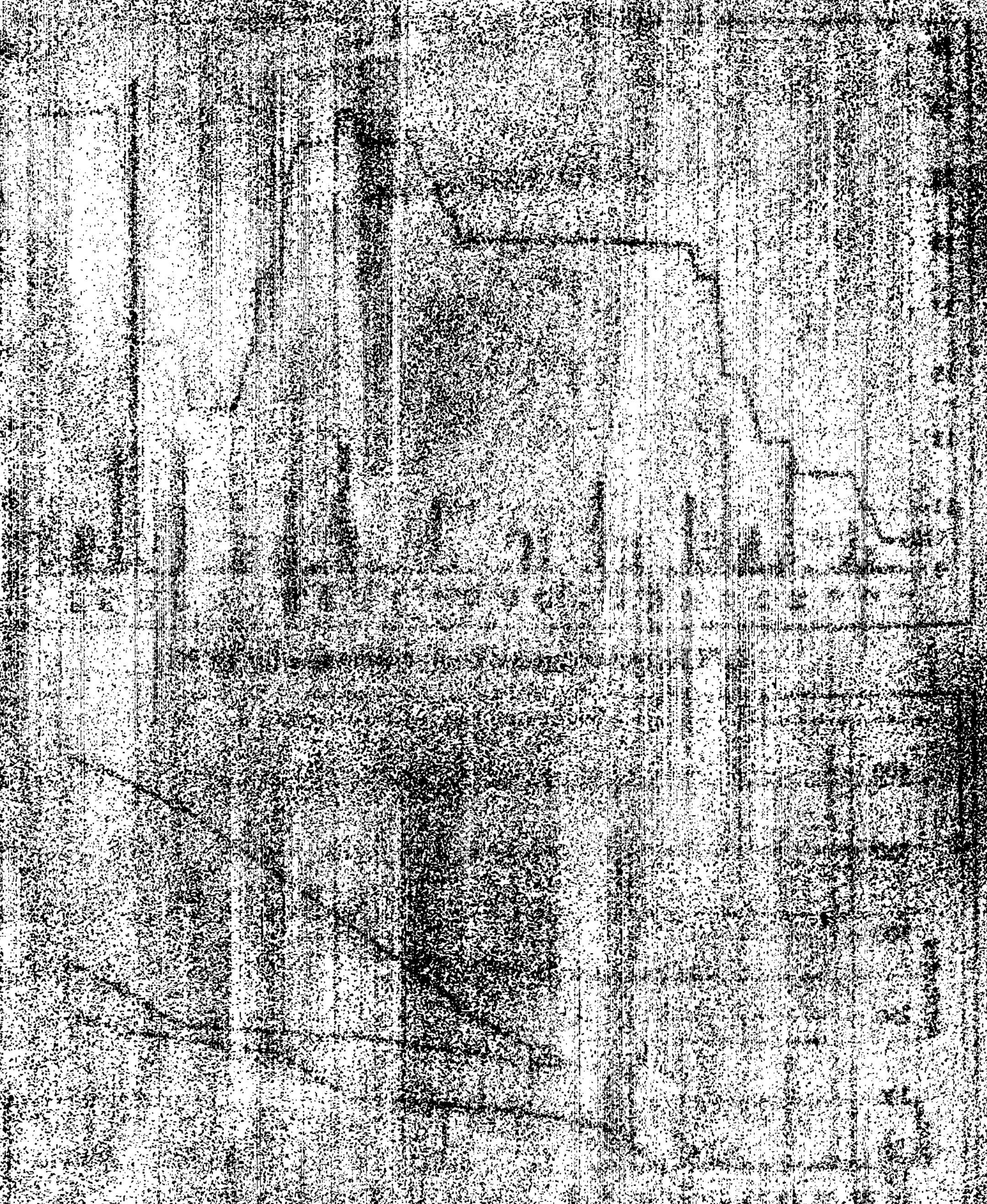


a) Changes compared with schema size for TYPO3



b) Timeline for TYPO3

Figure 3.24 Further insights for TYPO3



CHAPTER 4. ASSESING LEHMAN LAWS ON THE EVOLUTION OF DATABASES IN OPEN-SOURCE SOFTWARE

Lehman, in the seventies introduced a set of rules, or laws, on the behavior of software as it evolves over time. The laws of software evolution were amended, enriched and corrected by Lehman and his colleagues in the next thirty years [BeLe76, Lehm96, Leh+97, LeRP98, RaLe00, LeRa01, LeRa06] to form a body of eight rules known both as *laws of software evolution*, and, as *Lehman's laws of software evolution*. The first three laws were published in 1974, the next 3 six years later, and the final two laws in 1996.

In Table 4.1, we repeat the definitions of the laws: on the left column, we give the laws as they are presented in [Leh+97] (slightly adapted with respect to [Lehm96]) – in a publication has absorbed the evolution of the laws themselves and summarized them in a concise form for the first time, around thirty years after the initiation of this research line and with a the benefit of retrospect. On the right column, we give the laws as they appear in [LeRa06]; in this case, the effect of ten more years of research is apparent, as the laws appear in a more abstract form, including corrections and updated intuition.

Table 4.1 Laws of Software Evolution as stated in [Leh+97] (left) and [LeRa06] (right)

I -- Continuing Change	
(1996) E-type systems must be continually adapted else they become progressively less satisfactory.	(2006) An E-type system must be continually adapted or else it becomes progressively less satisfactory in use.
II -- Increasing Complexity	
(1996) As an E-type system evolves its complexity increases unless work is done to maintain or reduce it.	(2006) 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 -- Self Regulation	
(1996) E-type system evolution process is self regulating with distribution of product and process measures close to normal.	(2006) Global E-type system evolution is feedback regulated.
IV -- Conservation of Organisational Stability (invariant work rate)	
(1996) The average effective global activity rate in an evolving E-type system is invariant over product lifetime.	(2006) The work rate of an organisation evolving an E-type software system tends to be constant over the operational lifetime of that system or phases of that lifetime.
V -- Conservation of Familiarity	
(1996) As an E-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behaviour to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.	(2006) In general, the incremental growth (growth ratio trend) of E-type systems is constrained by the need to maintain familiarity.
VI -- Continuing Growth	
(1996) The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.	(2006) The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime.
VII -- Declining Quality	
(1996) The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.	(2006) 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 -- Feedback System	
(1996) E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.	(2006) E-type evolution processes are multi-level, multi-loop, multi-agent feedback systems.

4.1. Laws of evolution for software systems with a view to schema evolution

The terminology of the laws involves a classification of software systems in categories, out of which the single most important one is the class of *E-type systems*, which involve "software solving a problem or addressing an application in the real-world" [Leh+97]. The main idea behind the laws of evolution for E-type software systems is that the evolution is a process that follows the behavior of a feedback-based system. Being a feedback-based system, the evolution processes has to balance (a) positive feedback, or else the need to adapt to a changing environment and grow to address the need for more functionality, and, (b) negative feedback, or else the need to control, constrain and direct change in ways that prevent the deterioration of the maintainability and manageability of the software.

Before proceeding, we present a first apodosis of the laws, in our understanding, taking into consideration both the wording of the laws, but most importantly their accompanying explanations. We will revisit this apodosis later in this chapter, where we also discuss how we will measure it for the study of schema evolution.

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 (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 burst of effort from time to time (IV).

Now, we can visit each law on its own and discuss both its original intuition and its measurement concerning software systems as well as how we address the law in the context of schema evolution.

4.1.1. Continuing Change

The first law of software evolution is known as the law of "continuing change".

Definition. 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 environment evolves, the software that is intended to operate in the real world and 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 [XiCN09] include (a) the cumulative number of changes and (b) the breakdown of changes over time.

In our case, we assess the law of continuing change via the *heartbeat of changes over time and version* as our primary means of evaluation (secondarily, one can also study its breakdown to the type of changes).

4.1.2. Increasing complexity

The second law of software evolution is known as the law of "increasing complexity".

Definition. 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.

Lehman says that "this law may be an analogue of the second law of thermodynamics, or an instance of it" [Lehm96]. Remember that the second law of thermodynamics states (in one of

its many wordings) that "the entropy of an isolated system not in equilibrium will tend to increase over time, approaching a maximum value at equilibrium" – or in other words, an isolated system will tend to reach thermodynamic equilibrium (i.e., a state that experiences no change when isolated from its environment), and during this process, differences in temperature, pressure and density tend to even out (and this results in the halting of changes and the maximization of entropy)[1].

So, the main idea behind this law is that the evolution of software systems tends to increase their entropy (here: complexity), and therefore, in successful projects, effort to confront this increase, must be devoted. Moreover, when discussing the law, Lehman indicates the battle between two antagonizing processes over a fixed amount of resources for the maintenance of software [Lehm96]: 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 [LeRa06] puts it: "these activities have minor or no impact in functionality, performance or other properties of the software in execution". Here, we adopt the [SWBK] definition of perfective maintenance (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*".

A short discussion on complexity and its measurement. Since we will ultimately resort to measurements for verifying the law, before proceeding further, we need to confront a more 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 measureable properties of software. To give an example, Fenton and Pfleegler [FePf96] 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

[1] See http://simple.wikipedia.org/wiki/Second_law_of_thermodynamics
http://en.wikipedia.org/wiki/Thermodynamic_equilibrium
http://plato.stanford.edu/entries/statphys-statmech/

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 [LeRa06] 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 coming from electronic logs or version management systems. Therefore, approximations have to make. [ICMS09] lists a vast number of possible metrics to assess the validity of the law of increasing complexity that gives emphasis to structural complexity. The list includes metrics on programmer productivity, Lines-Of-Code, function complexity, cyclomatic complexity, coupling, etc. Lehman and Ramil [LeRa01], on the other hand, approximate complexity as the fraction of the increase in cumulative effort over the functional power of a system, with the latter approximated as the system's size. This fraction is a non-decreasing function over time, according to the second law. Simply speaking, under this interpretation of the law, complexity can be approximated as the ratio of (a) the effort spent between two versions over (b) the increment in the size of the system. Again, the problem is dereferenced on how to define metrics for these two entities, and most importantly, effort. Whenever data on person time are not available, *effort can be approximated* via the work-rate, which can in turn be approximated by the number of *modules handled*, i.e., the number of modules modified, removed or added to a system [LeRP98] or, the number of modules modified or added to a system [RaLe00].

Requirements and metrics for the assessment of the law's validity. Clearly, the levels of indirection and approximation to validate the law are too many. This has been recognized already in the literature. Interestingly, in [LeRa01] Lehman and Ramil state that the law can only be indirectly verified. In fact, in [LeRP98] the law is mostly verified by rationalization along with the existence of a regressive formula for the size of the system (law VIII). In [LeRa06] the validity of the law is further supported by the fact that the growth ratio declines over time (laws V and VI); this is attributed to the inevitable complexity increase that age brings to a system. So, overall, the law is actually indirectly verified in the literature.

On top of this, if we consider the intuition on perfective maintenance, we have to face the problem that it is very hard to isolate the actions that correspond to perfective maintenance only (either as entire releases or as part of the effort for a release). Only a precise documentation of activities can reveal this kind of information.

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 the once discussed in this second law of evolution.

Third, *we will attempt to approximate the measurement of complexity as the fraction of the evolution-affected relations (i.e., the number of relations modified, deleted or added to the schema) between two subsequent versions of the schema over the difference in the number of relations of the involved versions*. The main concept behind this formula is that, because we do not have a method to measure the internal complexity of the database schema, we try to estimate its complexity. 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. Assume we compare two transitions with the same denominator (i.e., difference in 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). The minimum value we expect to see is one, when the relations that we handled had the same amount of change in the database schema. As long as the effort in close to the growth of the schema, the complexity of the schema is low.

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

In the above setting, we opt to include the removed tables in the formula; removing a table from a database is a major modification (all code that refers to it crashes if anything) and such actions give a significant hint on maintenance actions. We should expect to see increasing complexity in "expansion versions" and drops in complexity (in perfective maintenance versions) from time to time. Again, we stress that this is simply a corroborative indication of the law's validity due to the several levels of approximation involved.

4.1.3. Self Regulation

The third law of software evolution is known as the law of "self regulation".

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) [Lehm97].

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 [Lehm97] calls "cyclic effects" and [LeRa06] calls patterns of growth.

Already in [Lehm97], experimental data suggested that the wording of the law was probably subject to amendments. In this previous form, the law stated: "The evolution process of E-type systems is self regulating with close to normal distribution of measures of product and process attributes." In its redefinition, the discussion on the normal distribution is removed and the authors generalize the notion of pattern growth to affect different kinds of properties like size, age, application area, team size, organizational experience or behavioral patterns. [XiCN09] also reports that small ripples do exist in the systems that the authors study, although the ripples are not equally distributed and positive adjustments (increase in growth) are more frequent than negative adjustments.

Metrics for the assessment of the law's validity. Typically the third law of evolution is measured via the measurement of system growth and the observation of ripples (peaks and valleys in the plot) [ICMS09]. For each transition, growth is defined as the difference between the new, target value and the old, source value of the transition. These ripples are assumed to indicate the existence and effect 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).

To assess the law for the database schemata that we study, we measure the *size of the system over versions and time* as well as the *growth of the schema in terms of number of relations and attributes.* We believe that the following criteria should hold for assuming the law as valid:

- Patterns in the size (number of relations) or growth (the difference in size between subsequent versions) of the system that indicate a 'mechanism' that produces a reoccurring phenomenon over and over in the lifetime of the database schema
- Some evidence of negative feedback; i.e., versions, or even periods, of perfective maintenance, mainly demonstrated by (i) the reduction of size, or, in a more loose requirement, (ii) growth lower than the "local" (last 5 or 10 versions) average growth
- Oscillations around the average growth – typically, in the literature, the observation that generated the birth of the third law was that growth oscillates around an average

value, and typically within a band of 2 standard deviations (practically resulting in a normal distribution of size alterations). Since new data indicate that growth declines with system age, we should expect to see the respective phenomenon with a declining band of values.

4.1.4. Conservation of Organizational Stability

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".

Definition. 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 decisions or 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). Thus, in its original form, the law stated that all taken into account, the rate of production is constant throughout the entire lifetime of a system. In its updated form, the law is refined to recognize the practical observation that the typical, predictable growth can be disrupted by abrupt changes that might be triggered by emergent external forces (e.g., mergers, downsizing, situations like the Y2K problem, etc).

Metrics for the assessment of the law's validity. As [XiCN09] excellently states, it is very hard to assess effort from the data that we can typical acquire from a project, as "effort does not equate progress". Therefore, we can only approximate effort by observing the published versions of a system. To this end, possible metrics [XiCN09] include (i) the number of changes per version, (ii) the average number of changes per day (by taking the amount of

changes between two versions and dividing by the time period) and (iii) the change and growth ratios.

In the case of the studied database schemata we will employ the following metrics:

- *Schema Growth*: As discussed in the previous law, growth is the difference between the size of the database schema in relations between the newest and the oldest database schema. More precisely growth = $size(v_{i+1}) - size(v_i)$ where size is the number of relations and v the version.
- *Heartbeat of Changes per Version*: The number of all changes between subsequent schema versions.

The goal is twofold: on the one hand, we need to detect whether there are phases with constant growth within them. On the other hand, we need to show that the phases are connected with each other via abrupt changes in this growth.

4.1.5. Conservation of Familiarity

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

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

In its previous form, the law stated: "During the active life of an evolving system, the content of successive versions is statistically invariant".

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 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"* [LeRa06]. 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.

[XiCN09, LeRP98] 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. [LeRa06] gives a large list of possible metrics: objects, lines of code, modules, inputs and outputs, interconnections, subsystems, features, requirements, and so on. [XiCN09] propose 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 adopt the same metrics for our studied database schemata and will also measure the system's growth via the following metrics:

- *Schema growth*: the net difference in schema elements of two subsequent elements, measured over (1.1) relations and (1.2) attributes.
- *Schema growth ratio*: schema growth divided by the size of the older version.

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. Primarily, we will produce a linear interpolation of the displayed metrics and see what the overall trend is.
- The second, side-effect 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.

4.1.6. Continuing Growth

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

Definition. 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 Lehman explains in [Lehm97] 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 [LeRa06] eloquently states "the former is primarily concerned with functional and behavioural 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 [XiCN09] 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-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 perform, as it appears impossible to automate it.

Thus, we resort to the approximation offered by (i) *the number of relations* and (ii) *the number of attributes* per version of each schema. Observe that we utilize the version id rather than time, as we are primarily interested to see growth between versions and not over time. If this numbers show an overall expansion trend over time (regardless of versions with maintenance that introduce occasional shrinking in size) then the information capacity of the database expands and the law holds.

4.1.7. Declining Quality

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

Definition. 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 ageing of the system, along with the increase in complexity, also calls for a reestablishment of assumptions and functionalities to serve the users' needs. [Lehm96] 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 [LeRa06], where the viewpoint on quality is generalized to all possible kinds of quality an organization might deem necessary: "The bottom line is that quality is a function of many factors whose relative significance will vary with circumstances. Users in the field will think of it in terms such as performance, reliability, functionality and adaptability. A CEO, at the other extreme, will be concerned with the contribution the system is making to corporate profitability, its market share, the corporate image, resources required to support it, the support provided to the organization in pursuing its business, and so on." Thus, since quality depends upon the viewpoint of users, managers, developers, each carrying his own interpretation and measures, it is only appropriate to let the involved stakeholders identify the aspects of quality that concern them and measure them accordingly. Under this interpretation, the law is general enough to subsume the second law, in our opinion. Quality involves both internal and external quality and in fact, any kind of quality aspect, as well as its assessment is left to be determined on a case-by case basis. [XiCN09] complement this viewpoint with a look upon internal quality of a system, referring again to all the metrics characterizing the complexity of a system.

Metrics for the assessment of the law's validity. Possible metrics [XiCN09] 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 [XiCN09] we are not really in a position to accurately measure external quality as perceived by the end users, the management, etc. This would require access to a detailed record of requests for corrections/additions of functionality and planned expansions that –to the best of our knowledge- is simply not possible to obtain from the repositories. We can approximate the internal quality via metrics that have been discussed in the second law, so this part is covered in law II. Overall, we will follow [LeRa01, LeRa06] and *presume that the law holds by logical induction, if it is strongly established that the laws of feedback (III, VIII) and the second law holds.*

4.1.8. Feedback System

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

Definition. 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" [Lehm96]. There is a big discussion in the literature on various components and actors that via their interaction limit and guide the possible ways via which the system can evolve. We refer the interested reader to [LeRa06] for this. From our part, we do not presume to accurately 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 whether this is a regressive mechanism whose behavior can be generally estimated.

Metrics for the assessment of the law's validity. We will perform *regression analysis* to *estimate* the size of the database schemata. Thus we need a formula that estimates the number of relations for each version of the schema. We adopt the formulas found at [Leh+97] and [LePr98] 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

$$\hat{S}_t = \hat{S}_{t-1} + \frac{\overline{E}}{\hat{S}_{t-1}^2}$$

where $\hat{S}$ refers to the estimated system size and $\overline{E}$ is a model parameter simulating effort (actually obtained as the average value of a set of past assessments of assessments of E). We discuss the different approaches that we have used for the validation of the law in the assessment part.

To assume the law as valid we need to establish that it is possible to simulate the evolution of the schema size via a formula that accurately follows the actual evolution of the schema size.

4.1.9. Putting it all together for schema evolution

Coming back to the problem under investigation, we need

(a) To deeply consider the fundamental philosophy of what the Lehman laws signify for the case of database schemata and schema evolution, and, consequently, how the laws are "translated" for the case of databases,
(b) to assess the applicability of Lehman laws to schema evolution, and,
(c) to perform this assessment it in a quantitative way with clear, measurable properties.

There are similarities and differences between the E-type systems that Lehman and his colleagues study compared to schema evolution, as studied here. Starting with the differences, we can observe the following fundamental differences:

- E-type systems export functionality to their users; on the contrary databases export information capacity, i.e., the ability to store data and answer queries. *Thus, we believe that when it comes to schema evolution, all references to functionality or functional capacity should be restated with a view to information capacity.*
- E-type systems are complete software systems that provide overall solutions to problems in the real world; databases, on the other hand, are typically parts of a larger information system, serving the purpose of accurately answering queries that populate the surrounding information systems with the necessary data. In other words, whereas

there is a holistic view of systems in the former case, we have a specific component of a larger ecosystem in the latter.

Having said that, however, we can counter this last difference by the following observation (that brings us in the area of similarities):

- Databases come with users having requirements from them (in terms of information capacity), developers and administrators that deal with them and the code that surrounds them, and thus, resemble typical software systems with a large degree of independence and a stand-alone character.
- Databases are fairly independent from the rest of the software modules and, thus, from changes to them. Typically, database evolution comes directly from requirements coming from the real world and not from the chain effect of changes within an information system. In fact, we can fairly say that whereas the rest of the software modules are typically dependent upon the database layer and impacted from its evolution, the inverse does not hold (or, at least, it holds very rarely). This practically presents the database with a peculiarity that simultaneously gives a similarity with E-type systems on the one hand and a difference on the other: *a database is a fairly independent module of an information system that is more or less insulated from changes to the other modules; at the same time, its evolution can potentially affect every other module.*

In the sequel, we restate Lehman laws adapted for the case of schema evolution, following the discussion of the previous subsections. We have grouped the laws in three categories according to their essence. The main concept of the laws is that the evolution of a software (in our case the database schema) is regulated by a multi-level, multi-loop, multi-agent feedback system (*VIII*). The positive and negative feedback loops and other control mechanisms (multi-loop) involve activities in many domains such as organizational, marketing, business, usage and so on (multi-agent, multi-level). This process drives forward (positive feedback) but also constrains (negative feedback) the evolution. A fundamental property of software systems (again, in our case, database schema) is that, as implied by the law of continuing change (*I*) the bounds of the feedback system governing their evolution change along with the system's

effort to adapt to environmental change but with a self-regulating manner (*III*). Thus, the first group, concerns the existence of a feedback system, and encapsulates the *VIII*, *I* and *III* laws. In terms of the properties of growth, i.e. the system's positive feedback, the need of the evolution to be driven forward is expressed by the law of continuing change (*VI*), which is regulated by the laws of conservation of familiarity (*V*) and organizational stability (*IV*), forming the second group. Finally, the negative feedback is expressed with the laws of increasing complexity (*II*) and declining quality (*VII*), which both imply the need of perfective maintenance. Considering the previous, we list the following hypotheses.

- *A database schema continuously changes over time (I^{db}) demonstrating a behavior that appears to obey a complex feedback-based evolution process ($VIII^{db}$) that can estimate its growth via a regressive formula. As with software systems in general, this is due to antagonism of positive feedback, requesting adaption and growth and negative feedback requesting control over change and code quality. The process is typically guided by a pattern of growth that demonstrates its self-regulating nature, including a smooth growth interrupted with versions dedicated to perfective maintenance (negative feedback) to calibrate the tendency for uncontrolled growth of the system's size and complexity (III^{db}).*
- *In the long term, we observe an increasing size (information capacity)-- in terms of relations and attributes-- of the database schema (VI^{db}), produced by a growth ratio that is slowly declining (V^{db}). The effort consumed for the above process is typically constant over phases, with the phases disrupted with burst of effort from time to time (IV^{db}).*
- *To regulate the ever-increasing growth, there is negative feedback in the system controlling both the overall quality of the schema (VII^{db}), with particular emphasis to its internal quality (II^{db}).*

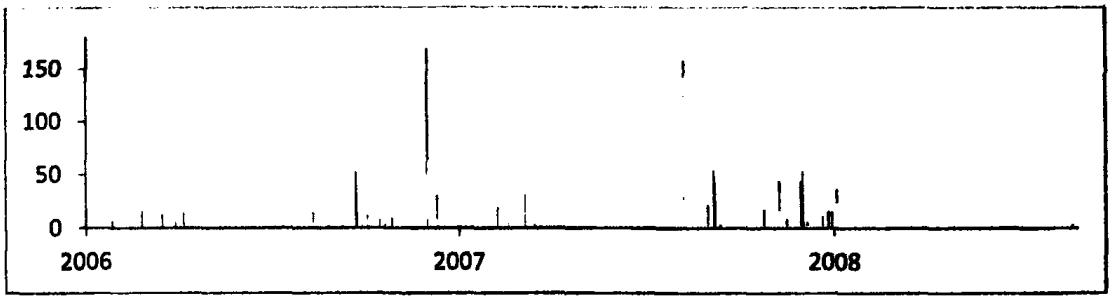
Table 4.2 Lehman laws adapted for the case of schema evolution and the metrics we used to access each law.

LAW	STATEMENT	MEASURES
I	*Continuing change* The database schema is continually adapted	We can accurately measure the law via heartbeat over time and version id
VIII	*Feedback System* We can estimate the growth of the schema size via regressive formula	We can assess the regressive formula of Lehman for size estimation with different alternatives for effort estimation
III	*Self-regulation* The schema growth comes in phases, each with smooth growth; each phase of the schema lifetime includes shrinking versions that calibrate expansive ones.	We can accurately measure the law via the schema growth and size for relations and attributes
VI	*Continuing growth* The database schema size is increasing in the long run	We can accurately measure the law via the schema size in terms of relations and attributes over version id
V	*Conservation of familiarity* The average growth between versions is slowly declining	We can accurately measure the law via its heartbeat the schema growth the schema growth ratio
IV	*Invariant work rate* The average activity rate is constant within phases of smooth growth that connect with bursts of effort	We can approximate activity by output and measure the law via: heartbeat over version id schema growth the schema growth ratio
II	*Increasing complexity* Efforts to maintain internal quality must be made	We can approximate the perfective maintenance activity via the schema size schema complexity
VII	*Declining Quality* Efforts to maintain internal and external quality must be made	We can conjecture the laws validity by logical proof.

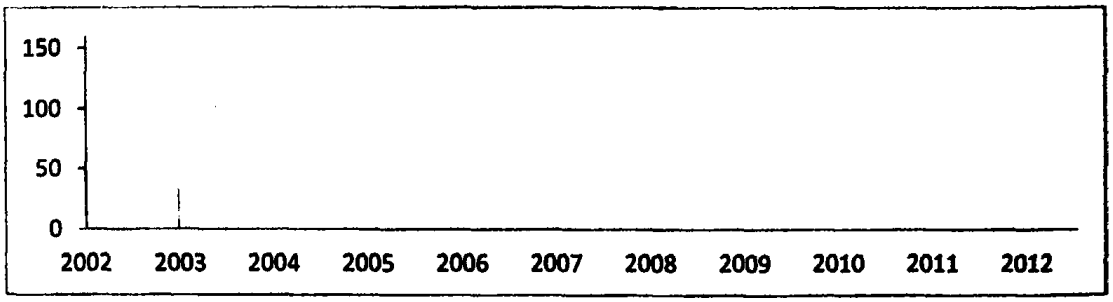
4.2. Measures of change and effort for Schema Evolution

In this section, we present the results for several measures that we monitored, assessed over the eight studied databases.

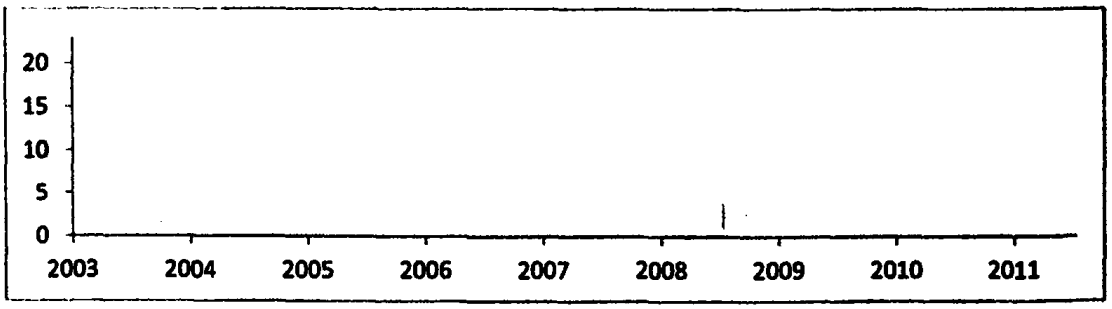
4.2.1. Heartbeat of Schema Changes over Time and Version Id



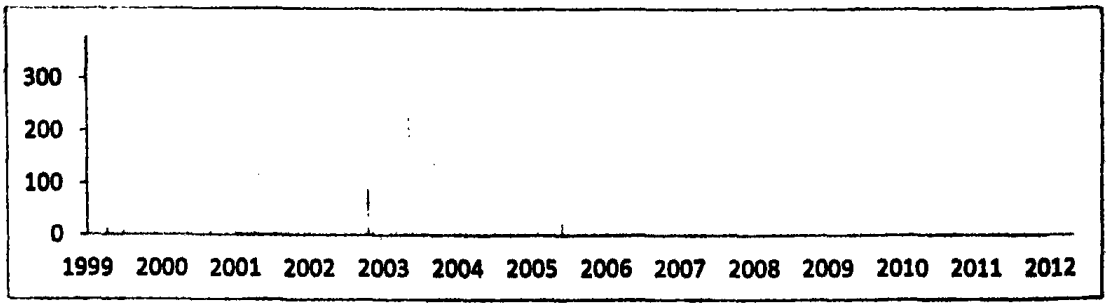
ATLAS Trigger



BioSQL



Coppermine



Ensembl

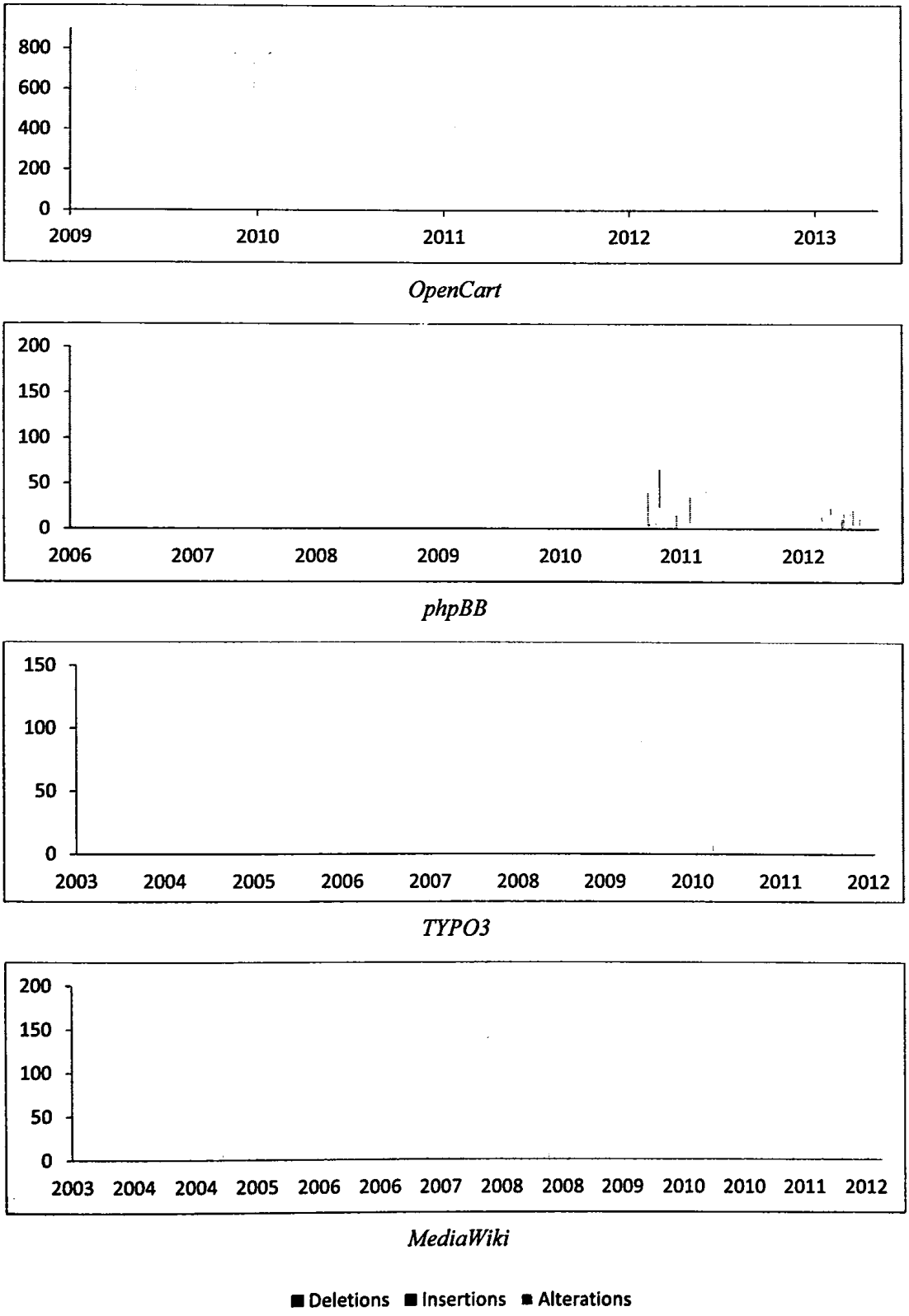
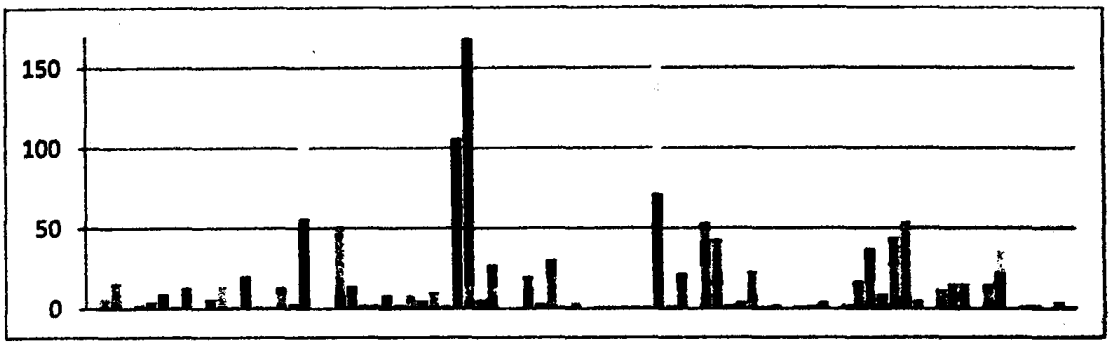
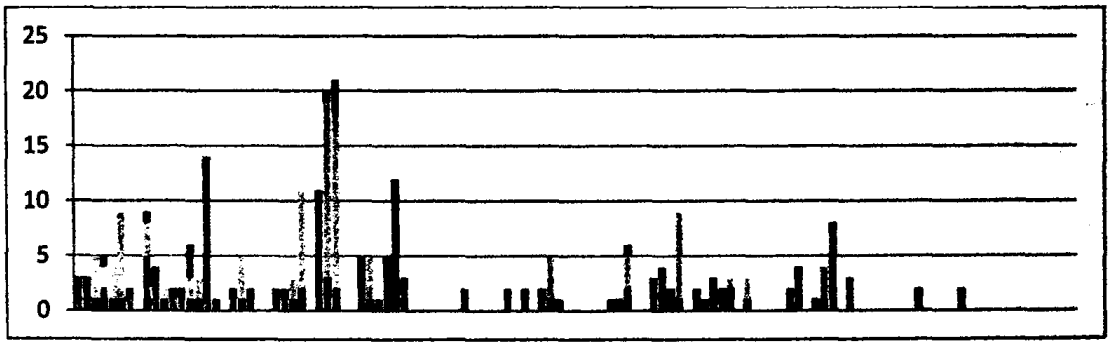


Figure 4.1 A comparative presentation of change breakdown -heartbeat- over time for the studied database schemata.

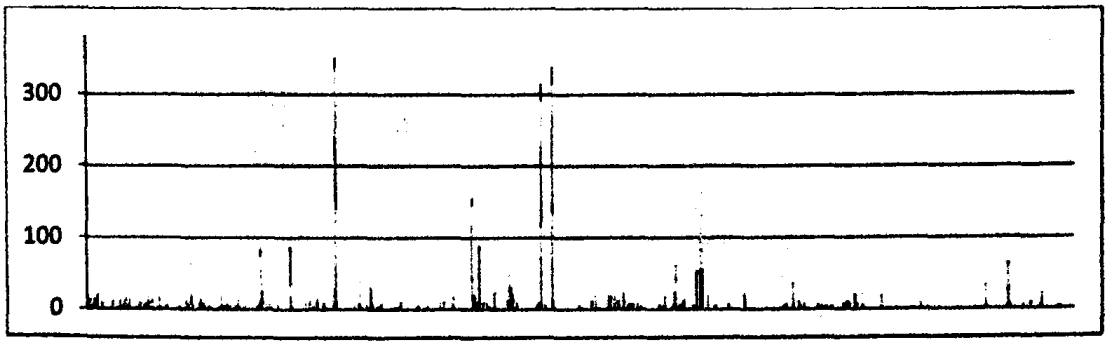


ATLAS Trigger

BioSQL



Coppermine



Ensembl

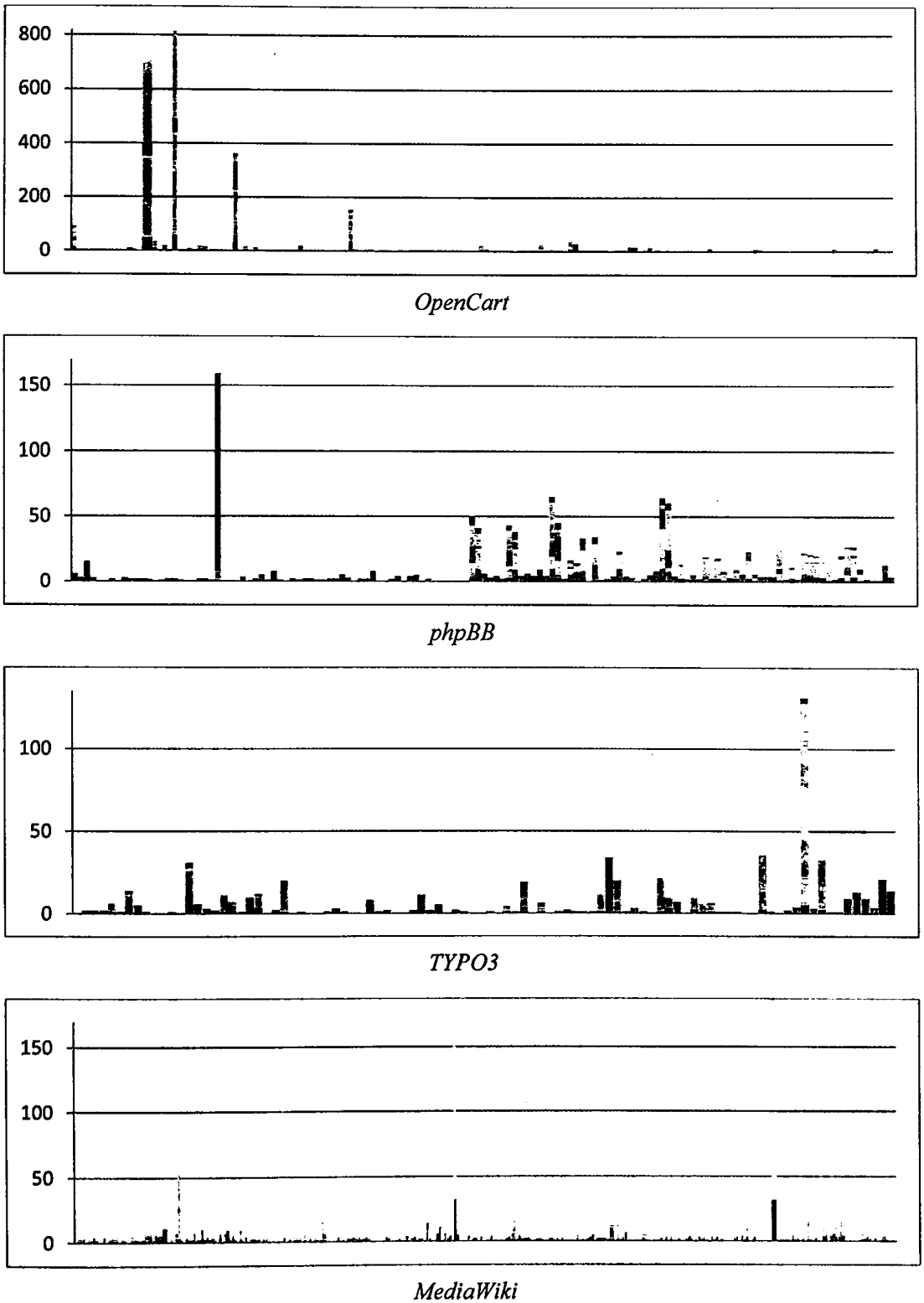


Figure 4.2 A comparative presentation of change breakdown -heartbeat- over version id for the studied database schemata

4.2.2. Schema Size (both in terms of relations and Attributes)

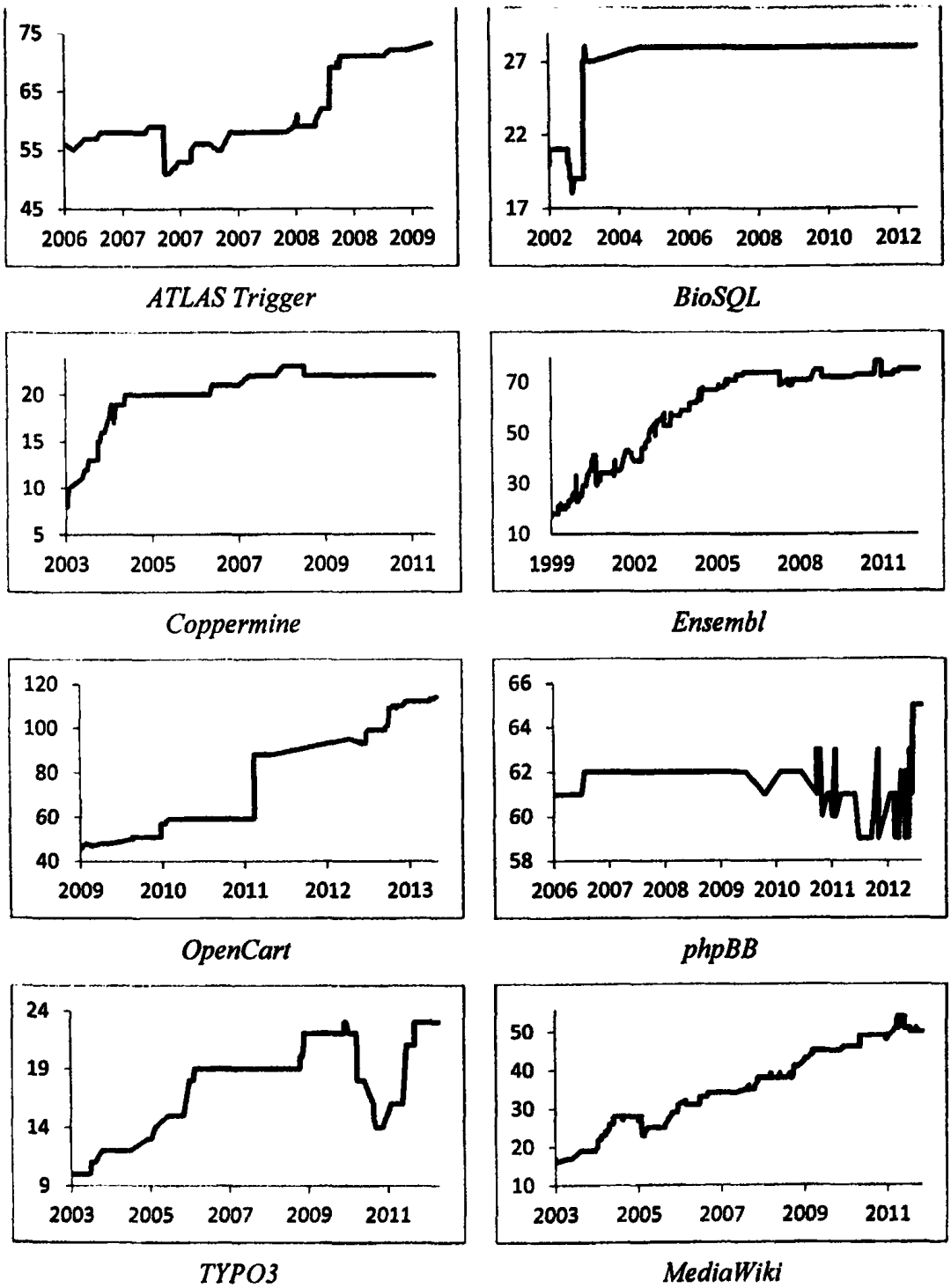


Figure 4.3 A comparative presentation of schema size per version over time, expressed as the number of relations, for the studied database schemata

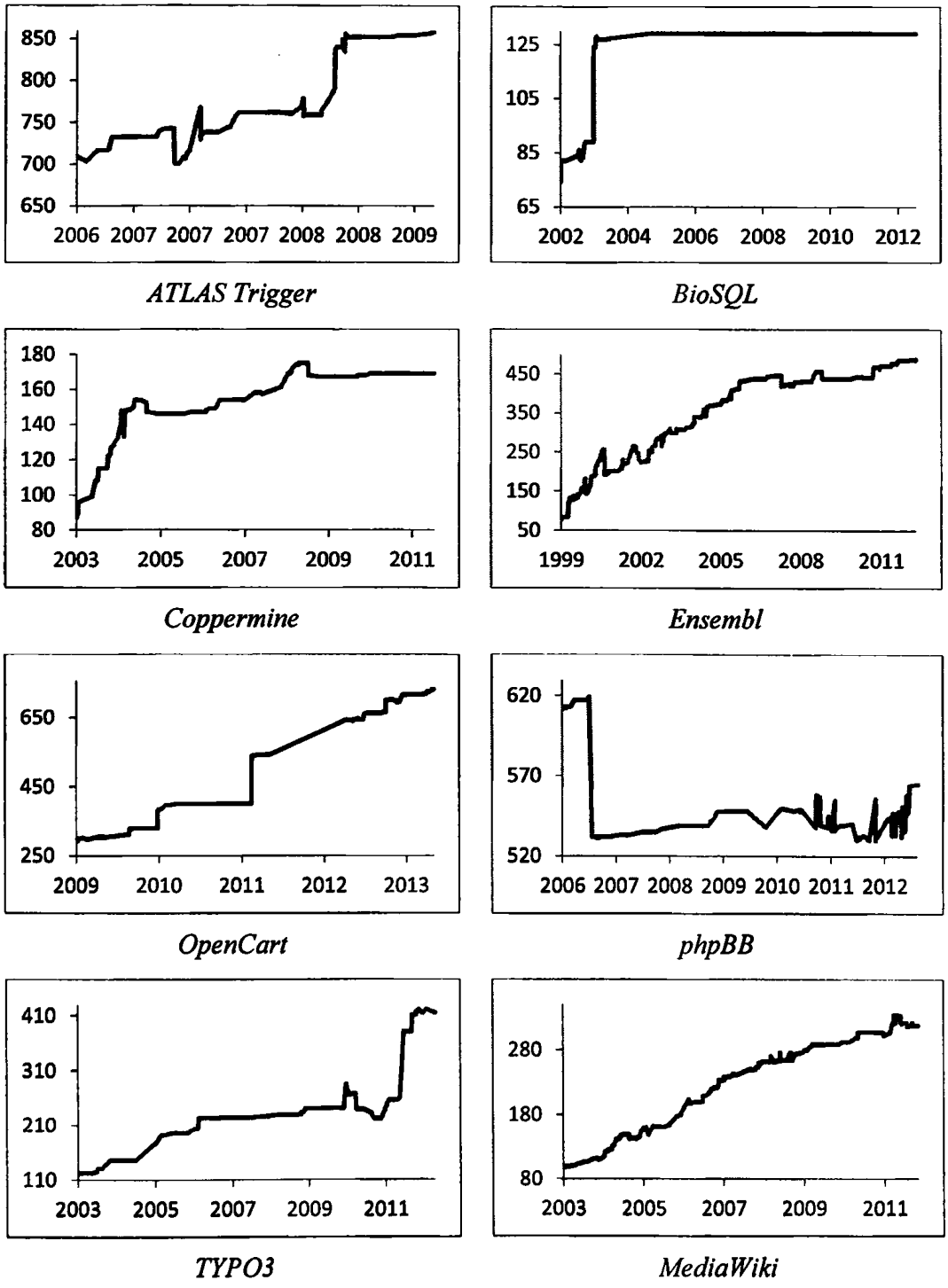


Figure 4.4 A comparative presentation of schema size per version over time, expressed as the number of attributes, for the studied database schemata

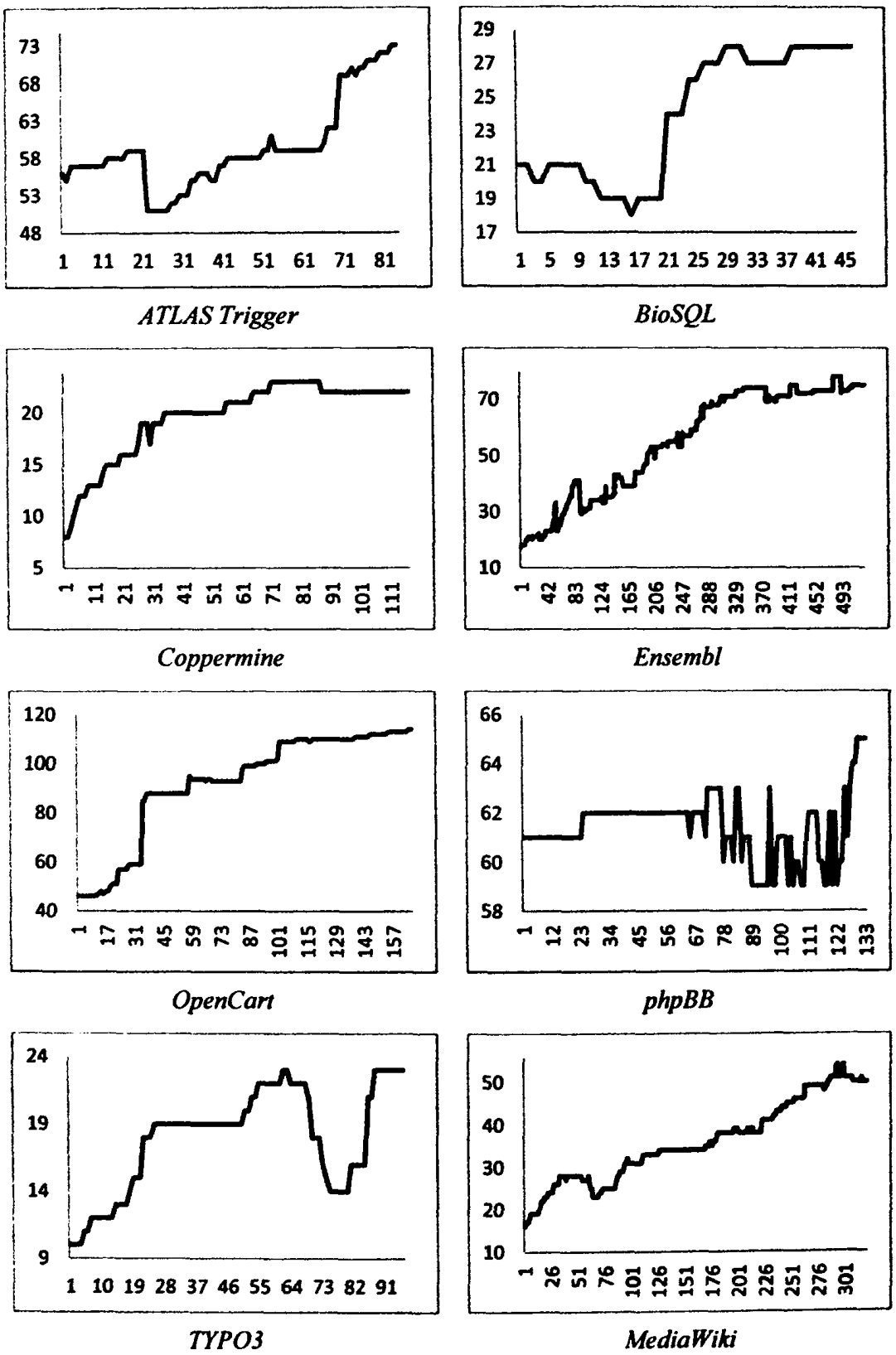


Figure 4.5 A comparative presentation of schema size per version, expressed as the number of relations, for the studied database schemata

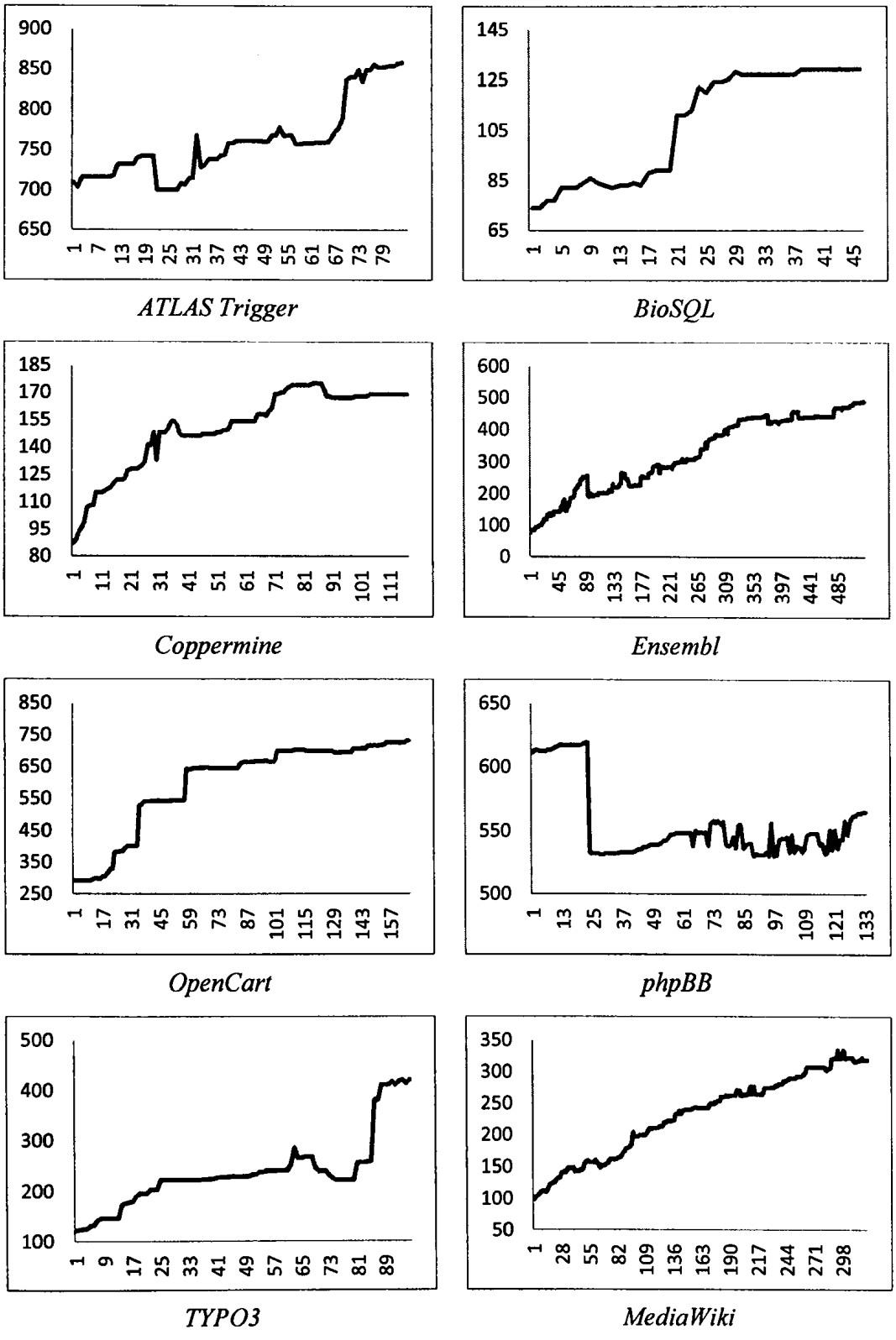


Figure 4.6 A comparative presentation of schema size per version, expressed as the number of attributes, for the studied database schemata

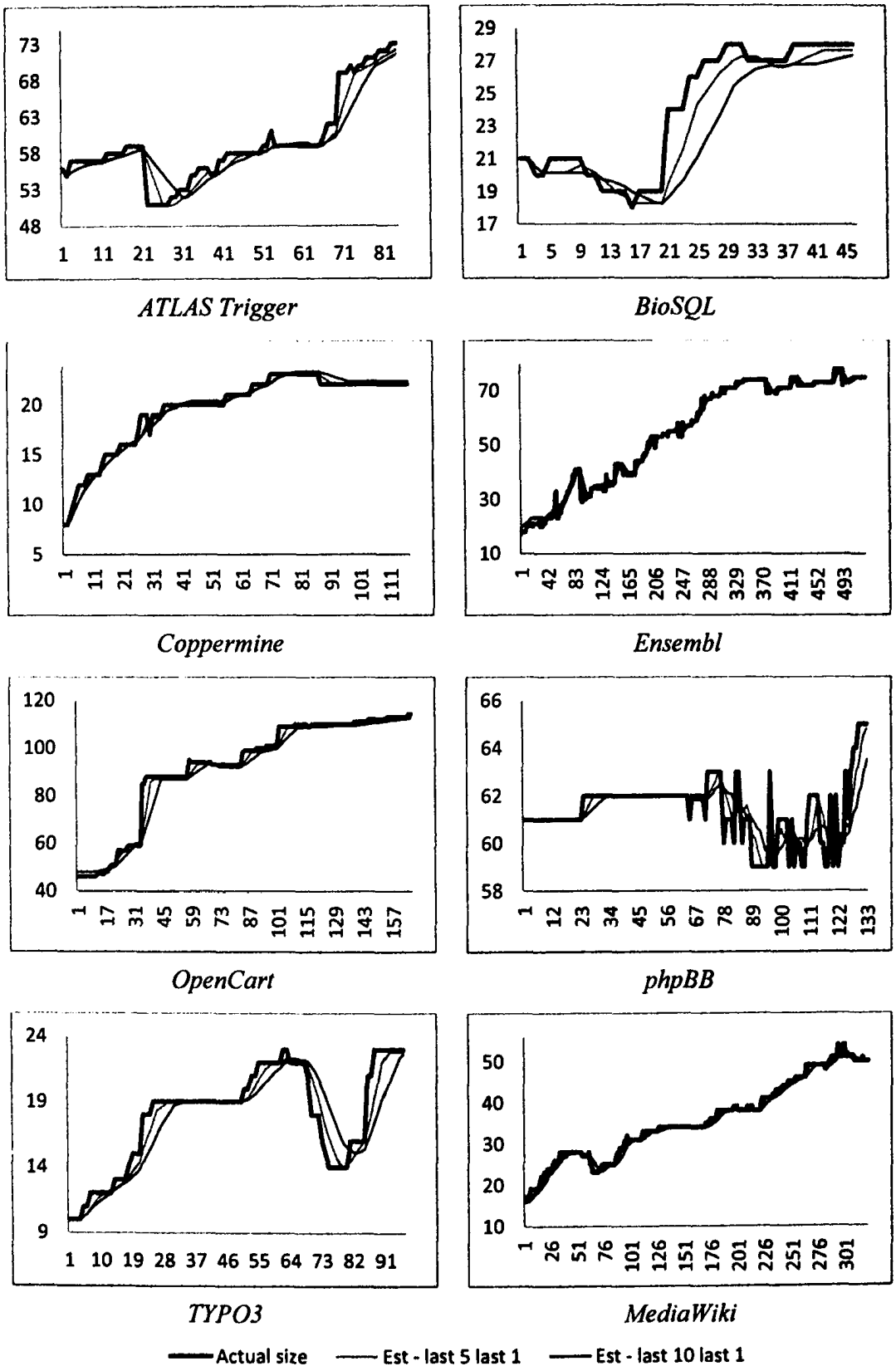


Figure 4.7 A comparative presentation of estimated size via regression analysis for the studied database schemata

4.2.3. Growth of the Number of Relations and Attributes

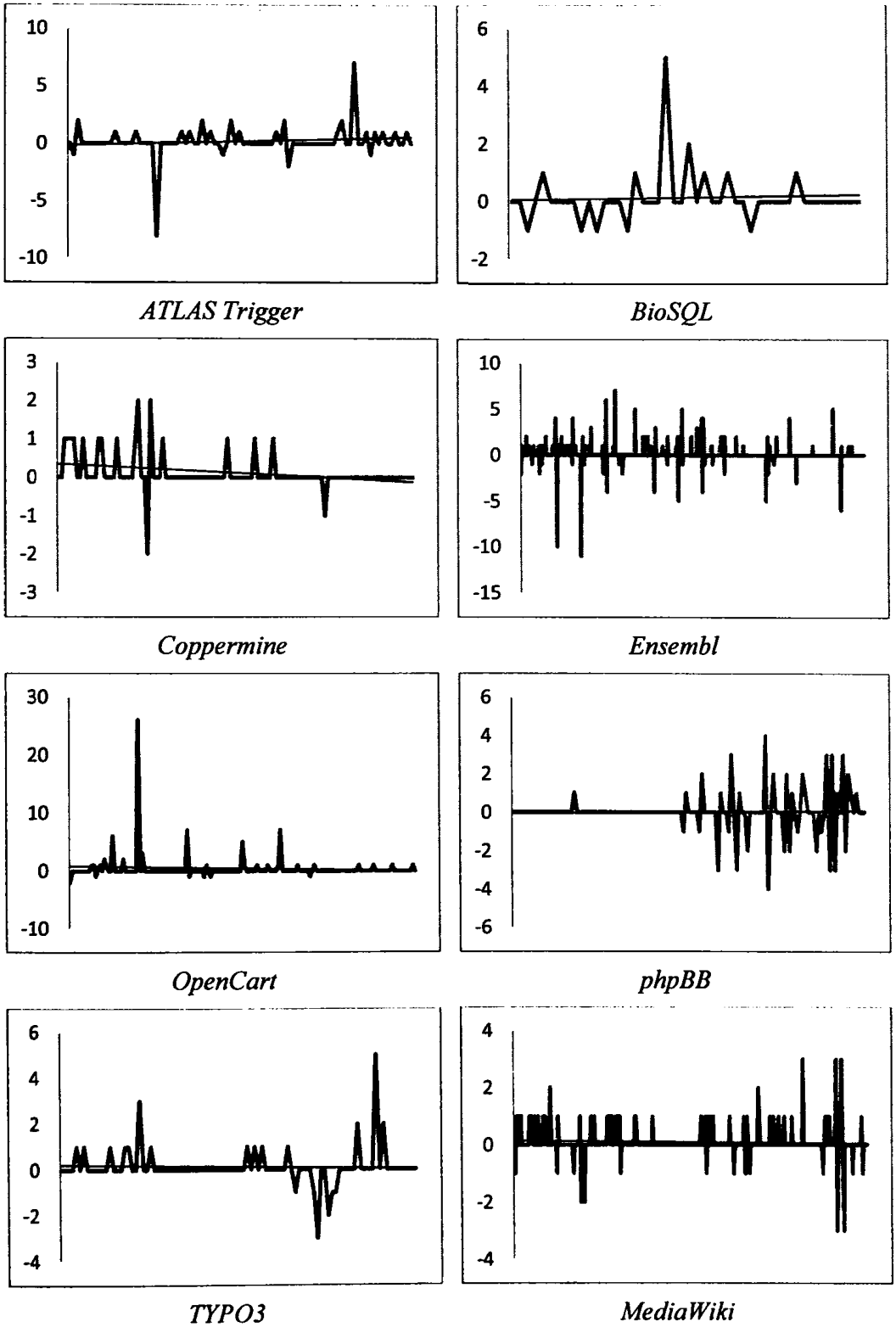


Figure 4.8 A comparative presentation of the growth (in number of relations) for the studied database schemata

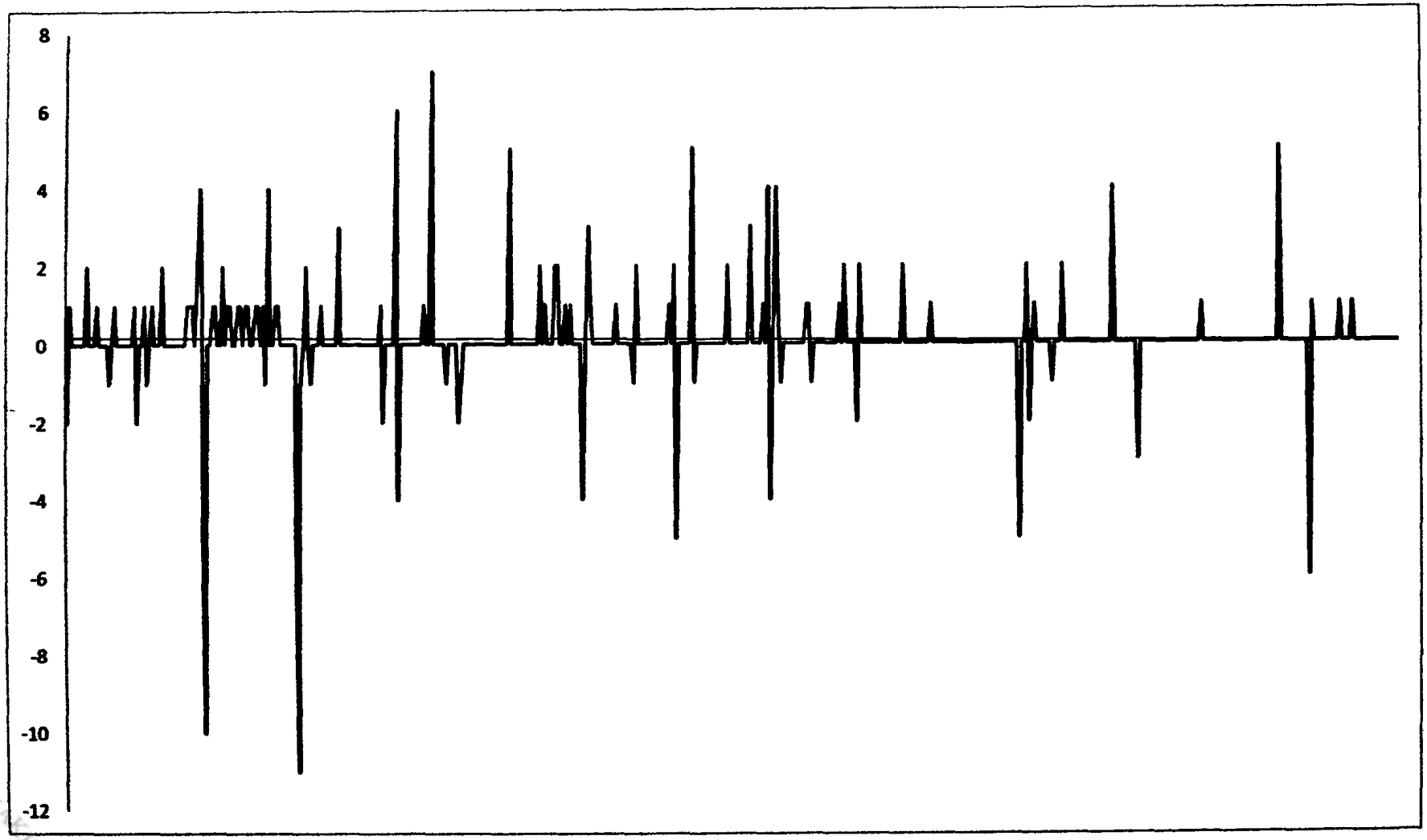


Figure 4.9 Zoomed growth (in number of relations) for Ensambl over version ID

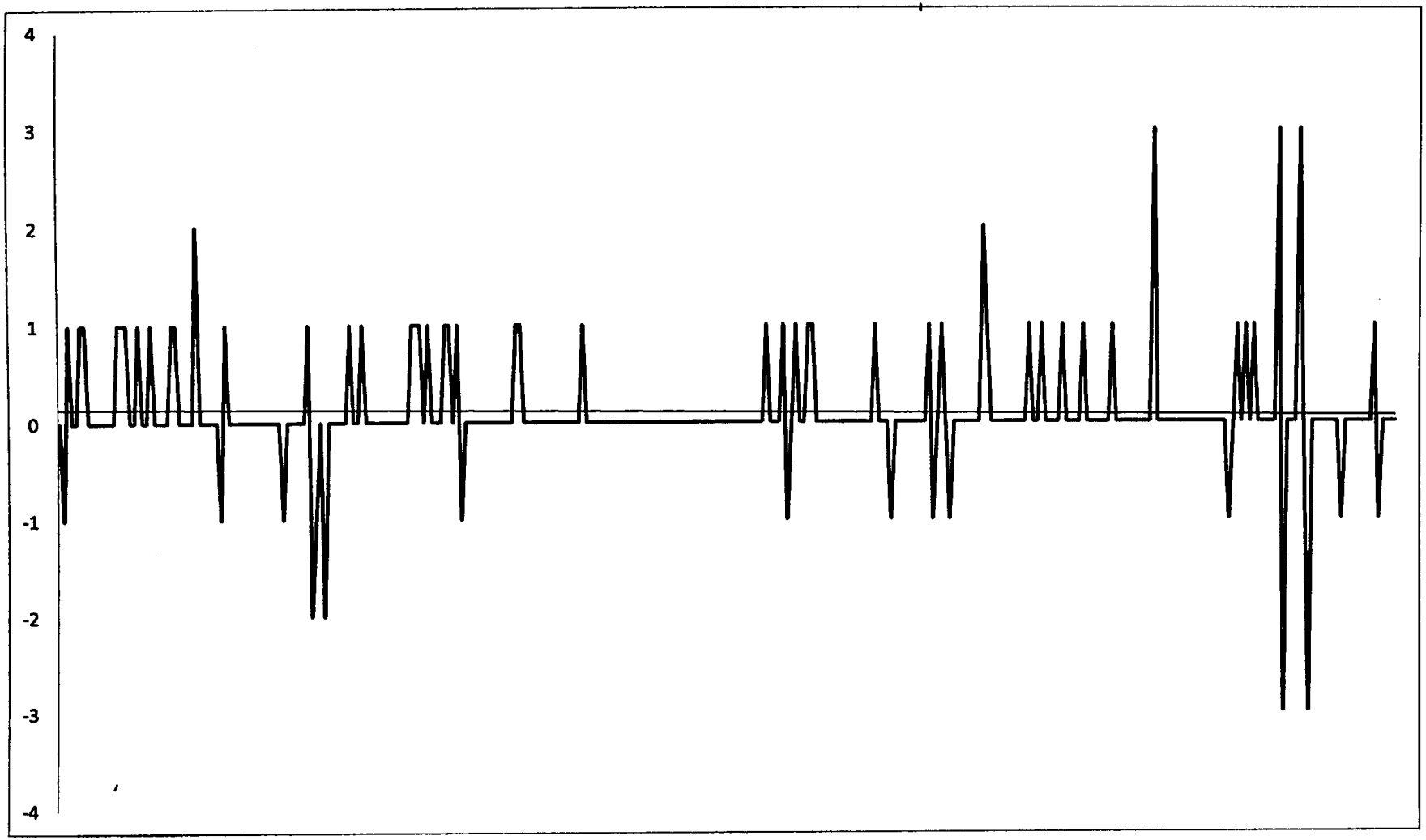


Figure 4.10 Zoomed growth (in number of relations) for MediaWiki over version ID

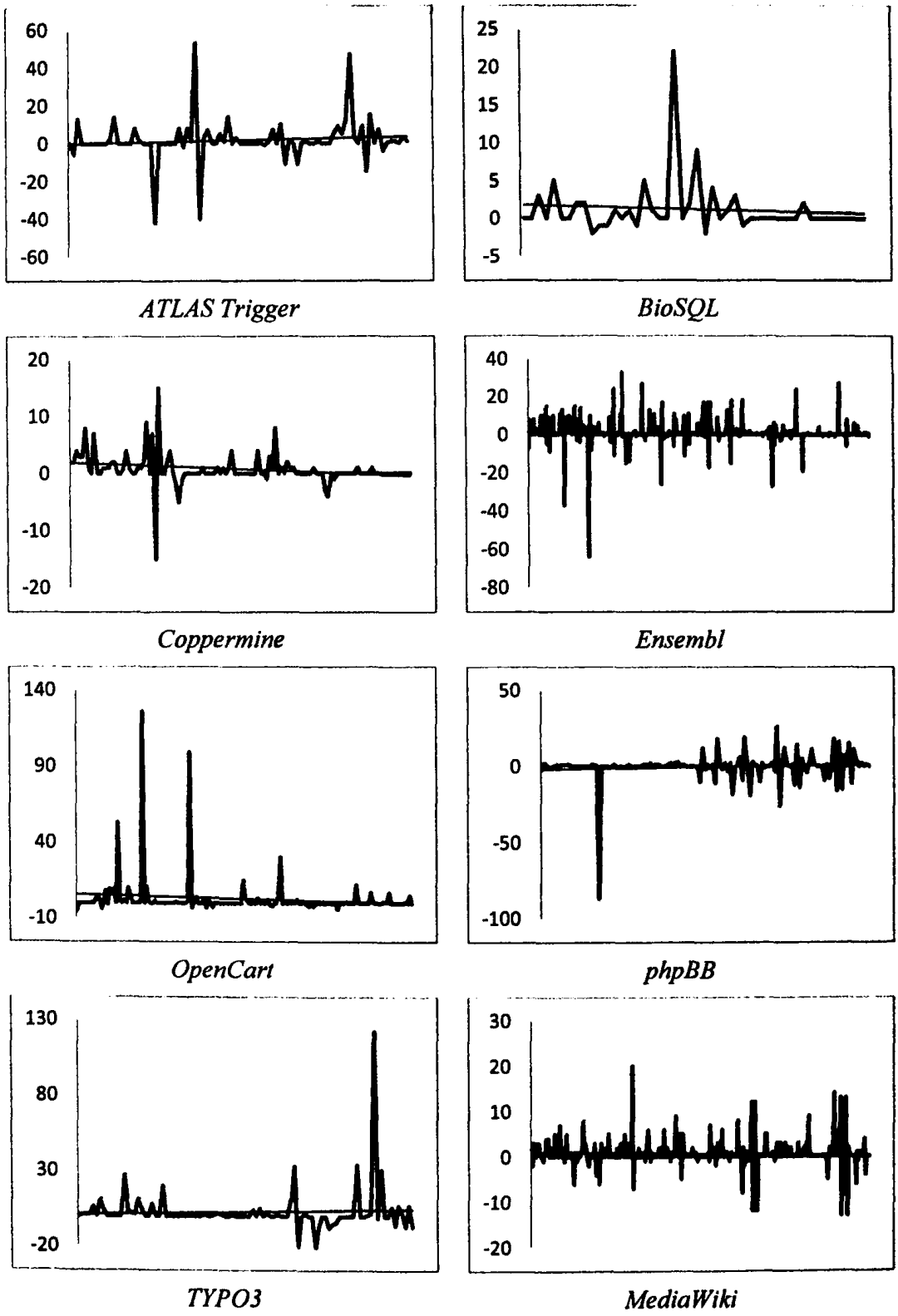


Figure 4.11 A comparative presentation of the growth (in number of attributes) for the studied database schemata

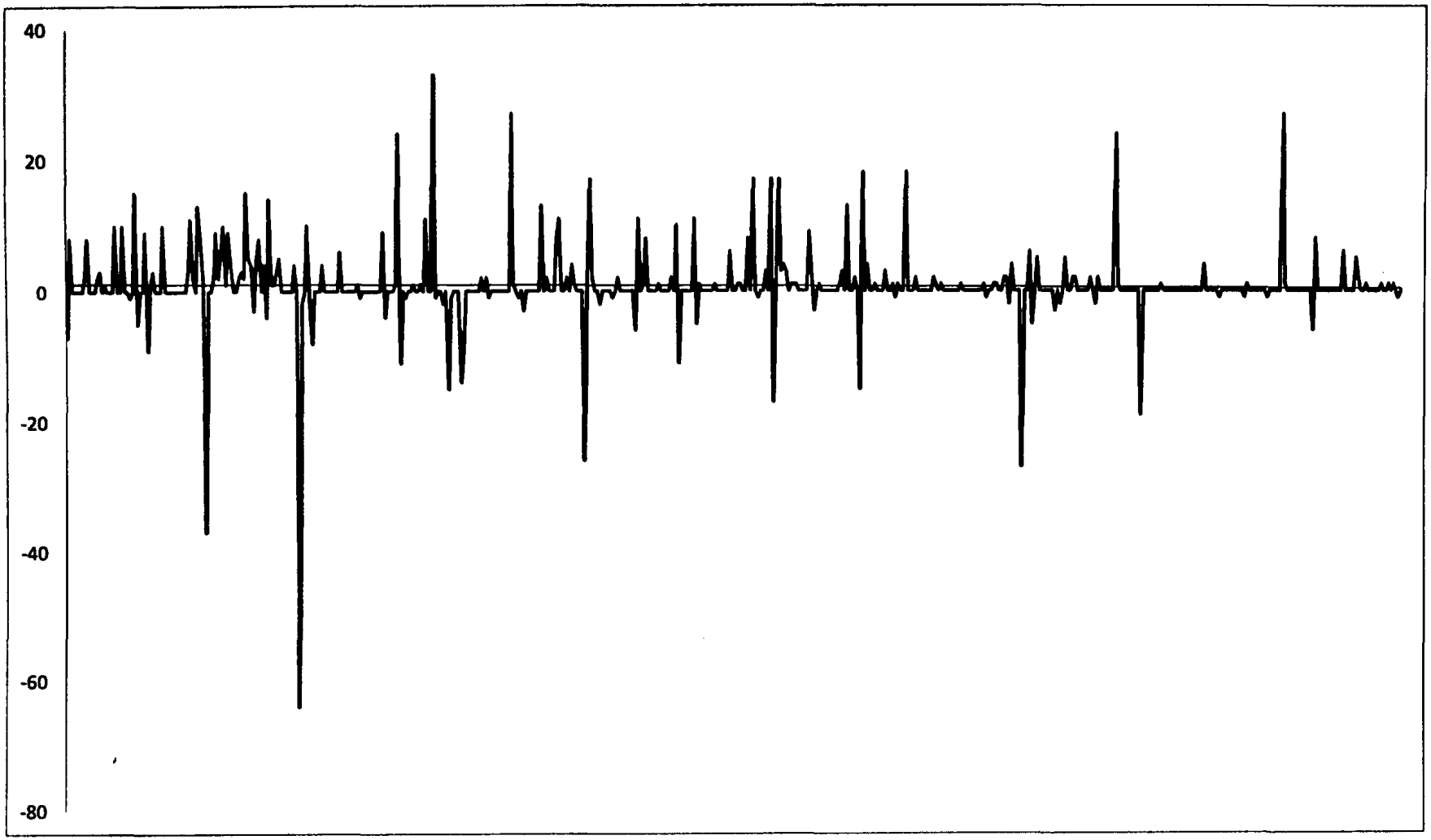


Figure 4.12 Zoomed growth (in number of attributes) for Ensembl over version ID

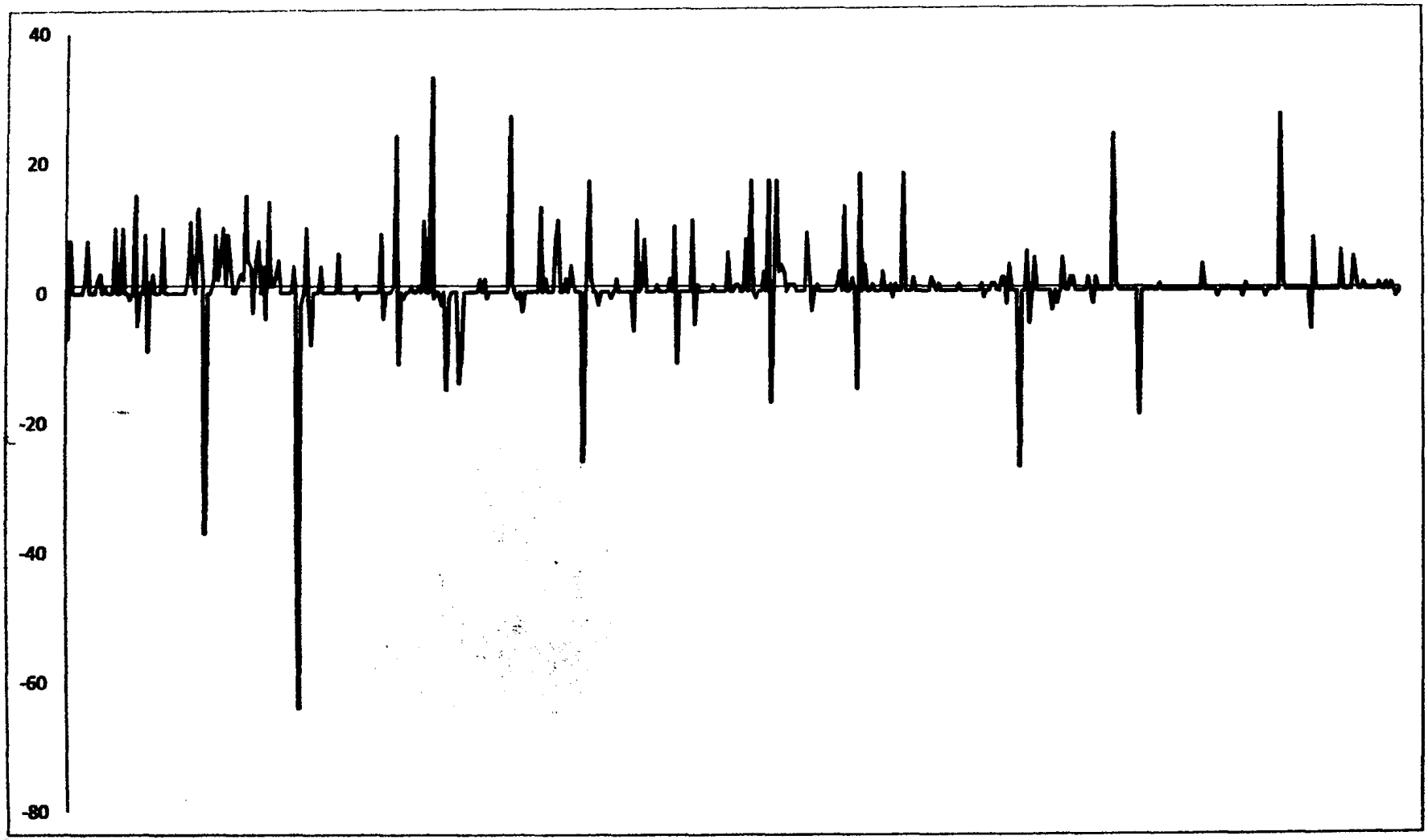


Figure 4.13 Zoomed growth (in number of attributes) for MediaWiki over version ID

4.2.4. Growth ratio

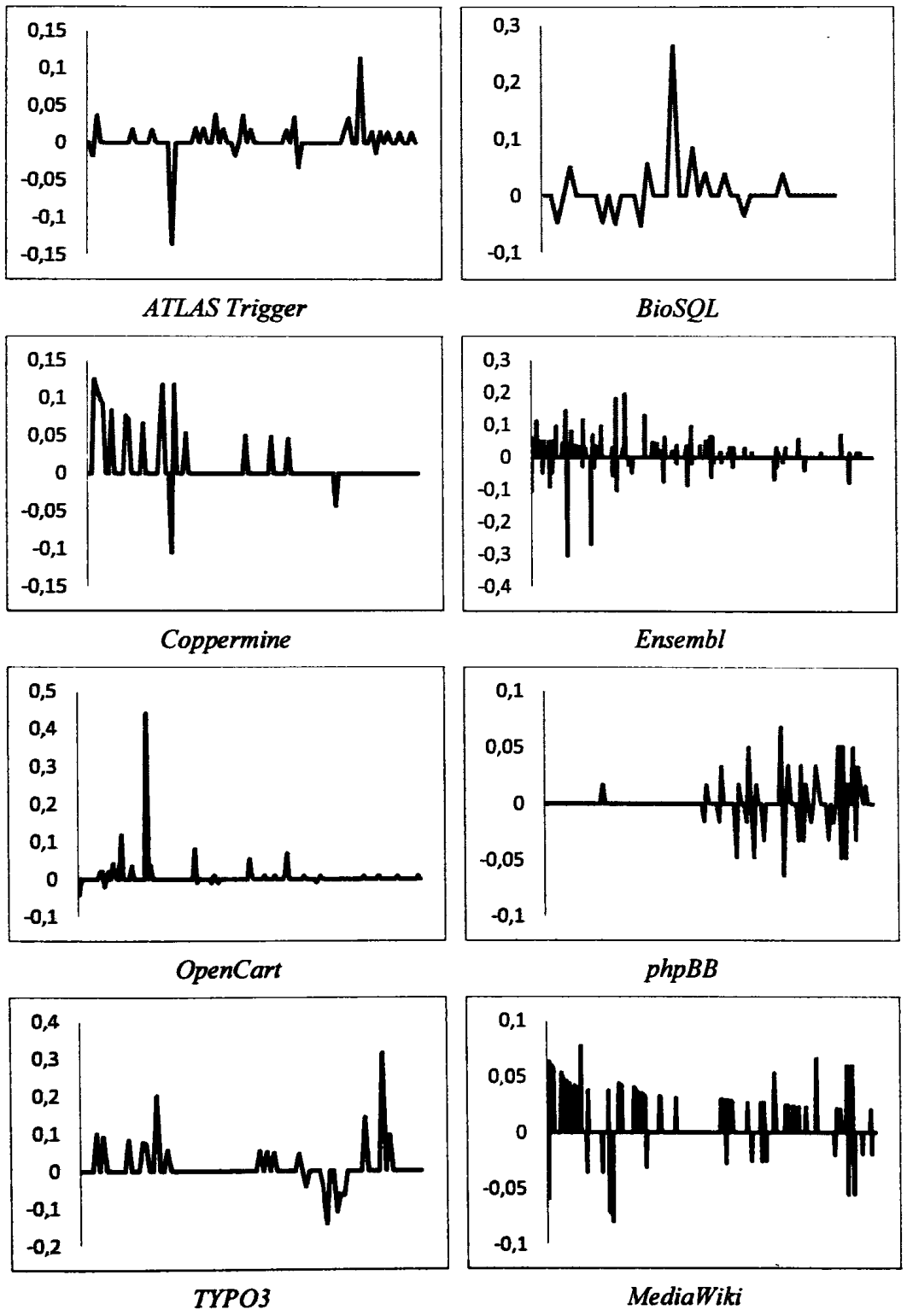


Figure 4.14 A comparative presentation of growth ratio for the studied database schemata

4.2.5. Complexity

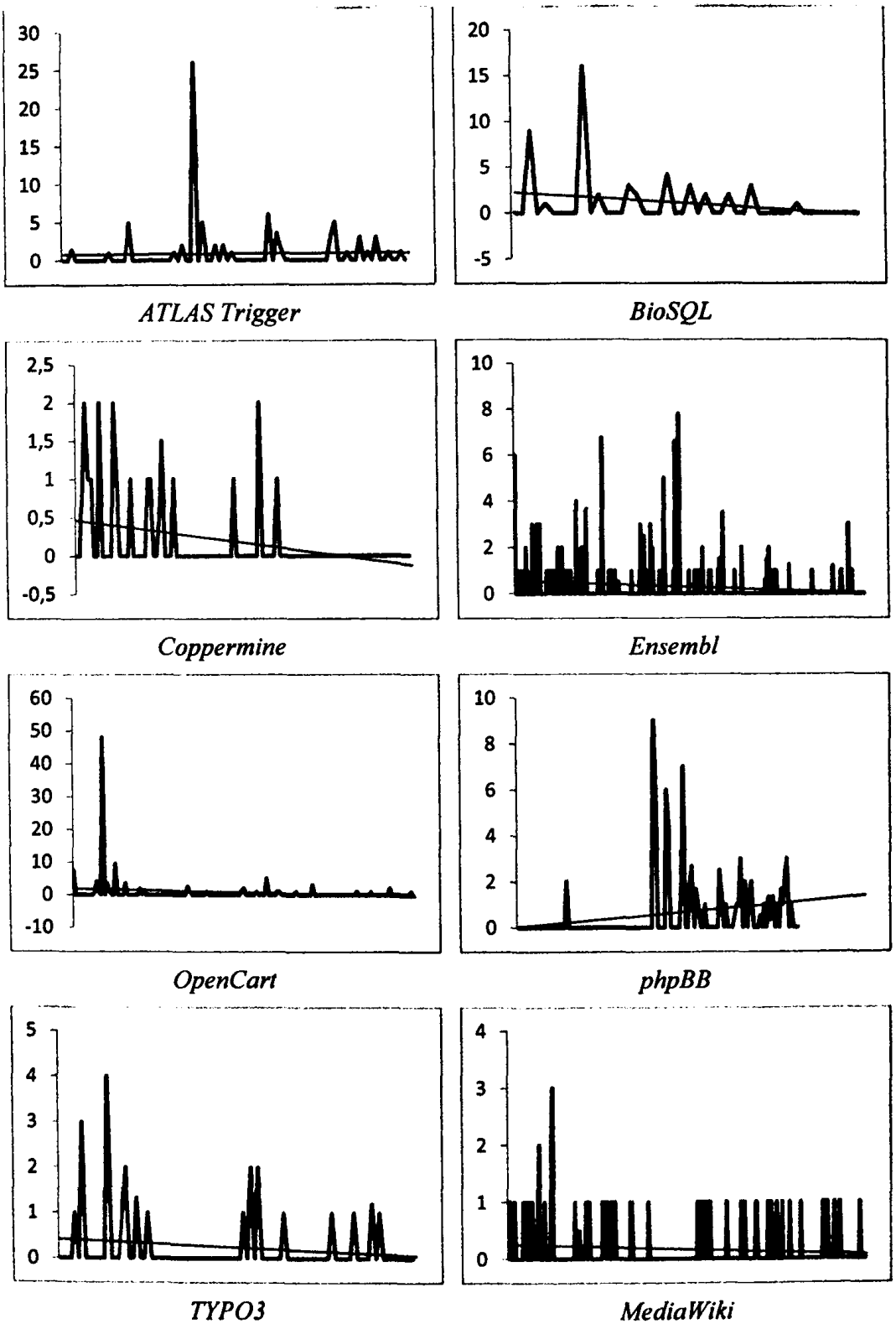


Figure 4.15 A comparative presentation of complexity for the studied database schemata

Having assessed the necessary measures of change for the eight schemata that we study, we are now ready to comment on our findings concerning the validity of Lehman's laws for the evolution of database schemata. We will construct our discussion around the three pillars of Lehman's argument: the feedback-based system, the positive feedback (the properties of the system's growth) and the negative feedback (the properties of the contraction of the growth).

4.3. Is There a Feedback-based System for Schema Evolution?

4.3.1. Discussion of our findings for the first law of continuing change.

The main argument of the first law is that the schema continuously changes over time. To validate the hypothesis that the law of continuing change holds, we need to study the heartbeat of the schema's life (over version id and time) and detect whether this is true.

We believe that the law of continuing changes *partially* holds for the case of databases for open-source software. In all cases (Figure 4.1) we see modifications (sometimes moderate, sometimes even excessive) over the entire lifetime of the database schema. Interestingly, there is also the exceptional case of BioSQL that appeared to be "sleeping" for some years and was later re-activated. But, for the vast majority of the studied schemata, the existence of changes throughout their lifetime is evident.

An important observation stemming from the visual inspection of our change-over-time data, is that the term "continually" in the laws' definition and name is challenged: *we observe (especially when the x-axis is time) that database evolution happens in bursts, in grouped periods of evolutionary activity, and not as a continuous process*! Specifically, the observation of the heartbeat over time heavily supports this statement; on the other hand, the inspection of the heartbeat over version id, comes also with stillness periods (or almost-stillness), but to a lesser degree. It is interesting to see that even if we study heartbeat over version id, we can still find versions with zero changes. As already mentioned, 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 refers to 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! Thus, evolution in databases is done with caution and care[1].

Overall, if we observe the exact wording of the law, we believe that the law partially holds. However, we should be cautious not to focus the evolution to the information capacity of the schema. We find the following wording appropriate (observe the removal of 'continuously').

From time to time, the database schema must be adapted to the users' information (logical schema) and performance (physical schema) needs or else it becomes unsatisfactory.

4.3.2. Discussion of our findings for the eighth law of feedback system

To validate the eighth law, we need to establish that the following formula can estimate the size of the schema (here: in terms of number of relations) accurately – i.e., with small error compared to the actual values. The formula is of the form:

$$S_t = S_{t-1} + \frac{\overline{E}}{S_{t-1}^2}$$

[1] On the other hand, there are cases like phpBB where this is not apparent.

where $\hat{S}$ refers to the estimated system size and $\bar{E}$ is a model parameter approximating effort (actually obtained as the average value of a set of past assessments of assessments of E). Once the formula has been introduced, the problem is now how to estimate $\bar{E}$.

Related literature [Leh+97, LeRP98] suggest computing $\bar{E}$ as the average value of individual E_i, one per transition. Then, we need to estimate these individual effort approximations. [Leh+97] suggests two formulae that we generalize here as follows:

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

where s_i refers to the actual size of the schema at version i and α refers to version from which counting starts. Specifically, [Leh+97] suggests two values for α, specifically (i) 1 (the first version) and (ii) s_{i-1} (the previous version, that is).

We now move on to discuss what seems to work and what not for the case of schema evolution. We will use the OpenCart data set as a reference example; however, all datasets demonstrate exactly the same behavior, as also depicted in Figure 4.7.

First, we start by trying to apply the formulae of [Leh+97]. In this case, we compute the average $\bar{E}$ of the individual E_i over the entire dataset. We employ four different values for α, specifically 1, 5, 10, and n, with n being the entire data set size, and depict the result in Figure 4.16, where the actual size is represented by the blue solid line. The results depicted in Figure 4.16, 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 $\bar{E}$ as the total average over all the values of the dataset, we compute it as the running average. This attempt was supposed to simulate the case where we do predictions and thus the running average is a good indicator. Again, the results are unsatisfactory.

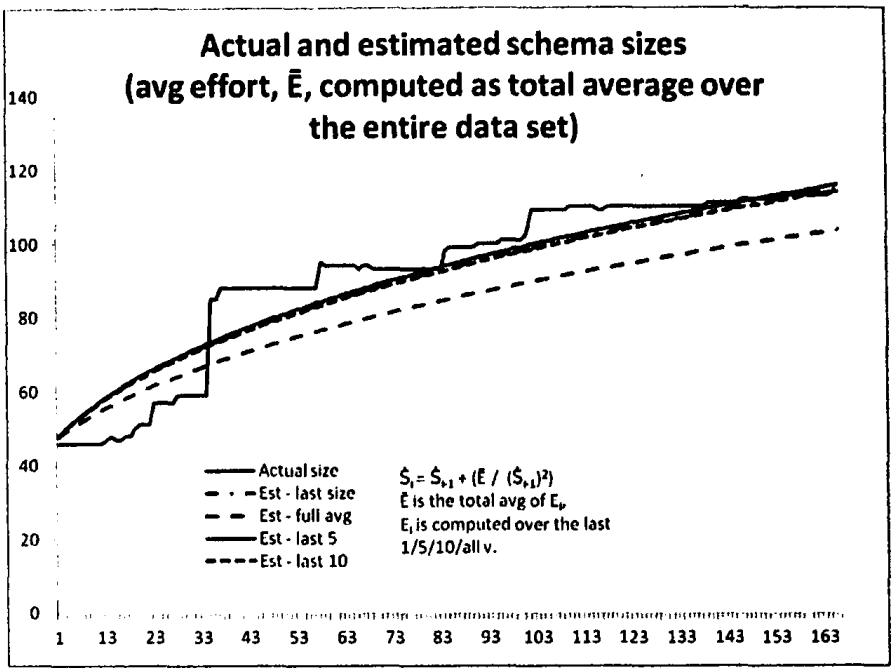


Figure 4.16 Actual and estimated schema sizes via a total average of individual E_i

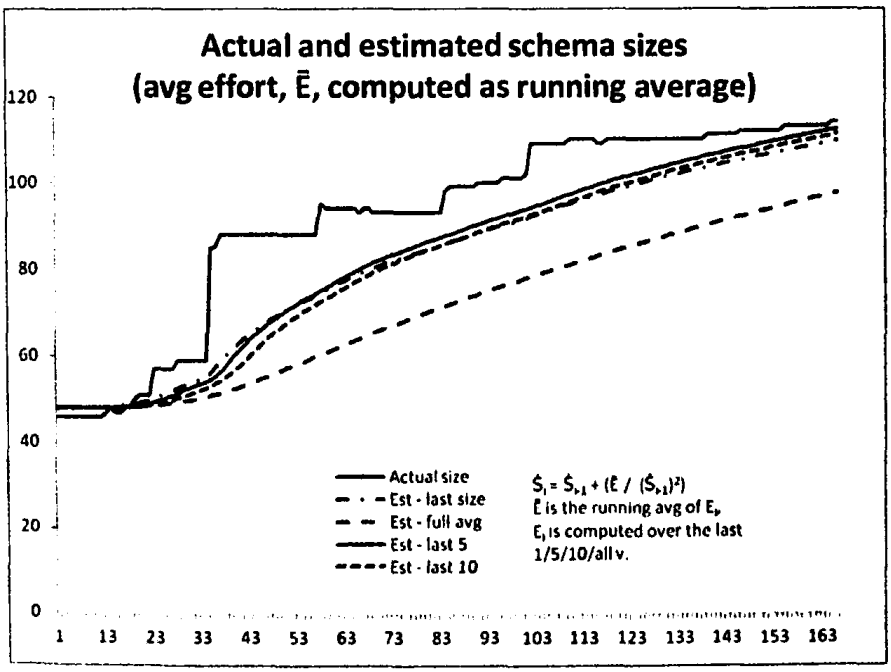


Figure 4.17 Actual and estimated schema sizes via a running average of individual E_i

After these attempts, we decided to alter the computation of $\bar{E}$ again. Our drive now was the observation that back in 1997-8 people considered that the parameter $\bar{E}$ was constant over the entire lifetime of the project; however, later observations (see [LeRa06]) led to the revelation that the project was split in phases. So, for every version *i*, we compute $\bar{E}$ 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 $\bar{E}$ 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.

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

As we can see in Figure 4.18, *the idea of computing the average $\bar{E}$ with a very short memory of 5 or 10 versions produced extremely accurate results. This holds for all data sets,* as depicted in Figure 4.7. This observation also suggests that, if the phases that [LeRa06] mentioned actually exist for the case of database schema, they are really small 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. In Figure 4.7, where all datasets are displayed, we depict only the two winner approaches, where the feedback's memory is very short, just one version back (here depicted as the blue and red solid lines "last 5 last 1" and "last 10 last 1").

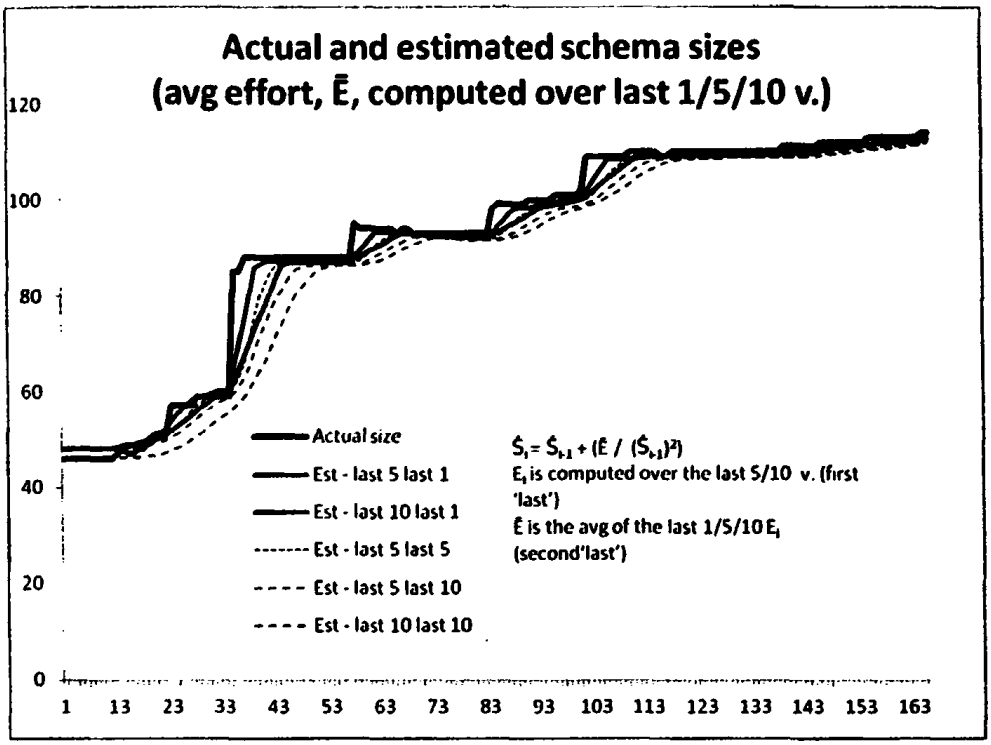


Figure 4.18 Actual and estimated schema sizes via a bounded average of individual E_i, also computed as bounded averages

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 $\hat{E}$. In plain words, this signifies that both size and effort approximation are intertwined in a multi-level feedback mechanism.

The evolution of the database schema appears to obey the behavior of a feedback-based system, 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.

4.3.3. Discussion of our findings for the third law of self-regulation.

To validate the hypothesis that the law of self-regulation holds, one expects to see the following phenomena: (a) recurring patterns in the schema size (in the area of software systems this is typically some form of a smooth expansion pattern) that produce a typical

"baseline" growth, (b) interrupted with releases of perfective maintenance with size reductions and/or growth, and, (c) with the growth demonstrating oscillations around the average value.

After observing the heartbeat of the schema both over time and over versions, for both relations and attributes (Figure 4.3, Figure 4.4, Figure 4.5 and Figure 4.6), we can make some interesting statements around each of the three studied phenomena.

Concerning the issue of a repeated, basic, fundamental pattern of smooth growth, interrupted with abrupt changes or, more generally, versions of perfective maintenance, we have to say that we simply cannot detect it in the studied database schemata. In sharp contrast to the respective figures of [BeLe76] and [Leh+97], the overall landscape of the size of the system provides a timeline with large variety of behaviors.

- In all schemas, there are versions with drops in schema size. Those drops are typically sudden and shift; usually they take place in short periods of time.
- In all schemas, 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 schemas, there are periods of stability (i.e., size stays still, or –near-still). If we observe the behavior of the relations and attributes over time, we frequently see stillness; in some schemas the stability remains for long periods of time. At the same time, the stillness is evident even if we observe the timelines having the version id as the x-axis. For relations, stillness is frequently evident in all datasets; for attributes we cannot really speak for stillness, but for stability or near-stillness, as the amount of change is typically small but not zero. However, if one observes the heartbeats over version id, for all schemata, one will always identify periods of non-modification to the logical structure of the schema.

Moving on to the second phenomenon, i.e., the existence of perfective maintenance, we can safely say that the existence of (large) drops in the schema size indicate the result of negative feedback, i.e. perfective maintenance. In fact, versions of perfective maintenance can be hidden in other versions, too, where we have renamings or restructurings of tables. As already mentioned, the automated identification of these phenomena comes with a degree of uncertainty that we have excluded from this study. Even without this information, however, there is still plenty of evidence to see that the schema is restructured from time to time resulting in (sometimes large) drops in its size.

Growth (i.e., the difference in the size between two subsequent versions) in all datasets has the following characteristics (Figure 4.8, Figure 4.11 and Figure 4.14):

- In terms of tables, growth is always very small (typically ranging within 0 and 1), and moderately small when it comes to attributes
- In terms of tables, we *have too many occurrences of zero growth*, typically iterating between small non-zero growth and zero growth. 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.*
- We also observe 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.
- 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).
- The growth ratio (i.e. the difference of sizes in a transition normalized over the old schema size) also comes with spikes and small values (typically ranging between 0% and 5%).

Overall, we can say that the essence of the law holds, despite the fact that we cannot confirm the pattern of smooth growth interrupted with releases of perfective maintenance based on our observations.

Schema evolution is feedback regulated as the existence of perfective maintenance is evident in the overall process. The growth comes with oscillations around the average value of growth which is close to zero (on the positive side) due (a) to the small size of delta's, (b) to the spike pattern (small change followed by no change) of growth, and, (c) the existence of several maintenance actions that balance the expansion of the system.

Thus, although the essence of the law, concerning the existence of a feedback mechanism, is still present in the case of schema evolution (making the law to hold, due to its very general definition), the mechanics of this system are clearly different than in the case of software systems.

4.4. Properties of Growth for Schema Evolution

4.4.1. Discussion of our findings for the sixth law of *continuing growth.*

The sixth law of continuing growth requires us to verify whether the information capacity of the system continuously grows. To this end, we study the size of the schema (over version id) both for relations and attributes (Figure 4.3). In all occasions, the schema size increases in the long run. We frequently observe some shrinking events in the timeline of schema growth in all data sets. However, *all data sets demonstrate the tendency to grow over time*. Concerning the number of attributes, this is also confirmed in all but one case (phpBB, which is clearly an outlier with respect to the rest of the datasets, starts with a high number of attributes, quickly drops sharply and then grows with ups and downs).

In both charts for the number of relations and the number of attributes, we see the schema growth to pass from three kinds of phases:

- *Stability*, which is the term we use to refer to absolute stillness or near-stillness (small changes)
- *Smooth expansion*, which refers to sequences of version where the size grows monotonically

- *Abrupt change*, where the size changes significantly (either positively or negatively)

The above observation holds both for relations and attributes. In contrast with observations made by Lehman regarding this law, in addition to the two phases that Lehman has discussed (smooth growth and abrupt change); we also observe the stability periods that appear to be unique in the case of database schema evolution. This 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, albeit modified to accommodate the particularities of database schemata.*

As an overall trend, the information capacity of the database schema is enhanced

---

4.4.2. Discussion of our findings for the fifth law of *conservation of familiarity.*

The fundamental question around the fifth law 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? A second question, also of interest for the fifth law's intuition 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?" To answer these questions we observe the charts depicting the growth and the growth ratio of the schema, also with the aid of the heartbeat charts.

Again, 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).

Overall, we do not see a diminishing in the values of growth; what we observe, however, *is a reduction in the density of changes and the frequency of non-zero values in the spikes. This*

explains the drop of the average value in almost all the studied data sets (see Figure 4.8 and Figure 4.11: 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).

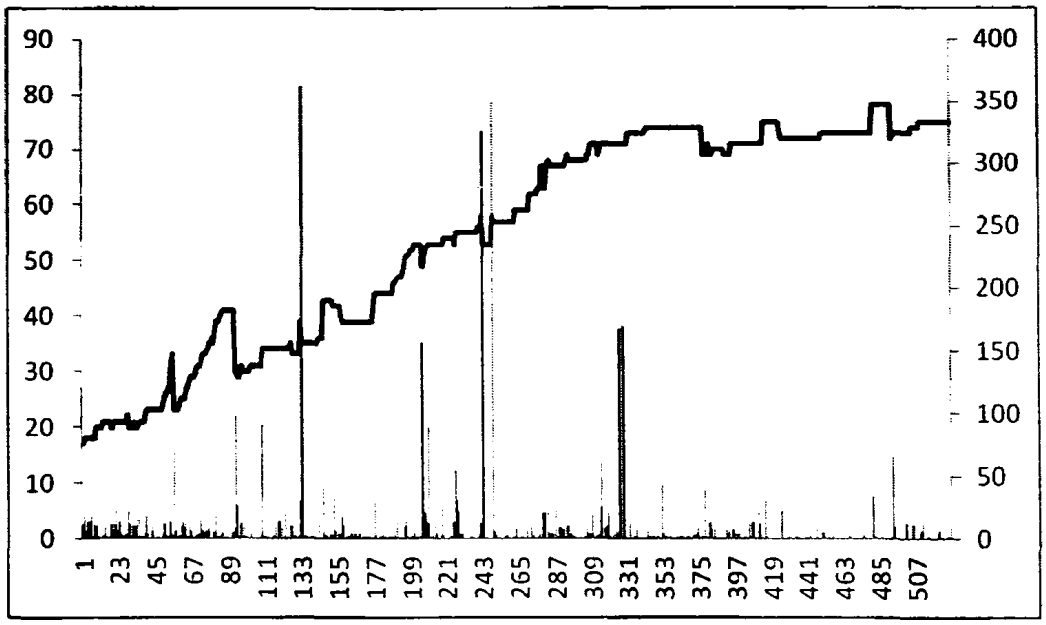


Figure 4.19 Ensembl's combined heartbeat and schema size: age results in a decline of both activity and growth

The heartbeat of the systems tells a similar story: typically, change is quite more frequent in the beginning, despite the fact that existence of large changes and dense periods of activities can occur in any period of the lifetime. Figure 4.19 clearly demonstrates this by combining schema size and activity. This trend is typical for almost all of the studied databases, possibly with the exception of TYPO3 and phpBB (as usual). phpBB demonstrates some increased activity in its last versions – we attribute this to the fact that apparently there was an expansive effort in this time period (around 2012) and will probably return to lower rates in the future versions. phpBB which is almost always an outlier, shows also some consistent activity at its later versions; still this involves a small amount of change that oscillates between 60 and 62-63 tables which is a very small difference actually (as the figures of phpBB are fitted to show the lines as clearly as possibly, they can be deceiving as to the amount of change -- the growth ratio is typically around 2%).

Concerning the validity of the law, we are not in position to verify the explanation of the growth. The law states that the growth is constrained by the need to maintain familiarity. Although this seems reasonable, we believe that the peculiarity of databases, compared to typical software systems is that there are other good reasons to constrain growth: (a) a high degree of dependence of other modules from the database and (b) an intense effort to make the database clean and organized. Therefore, conservation of familiarity, although important cannot solely justify the limited growth. The extent of the contribution of each reason is unclear.

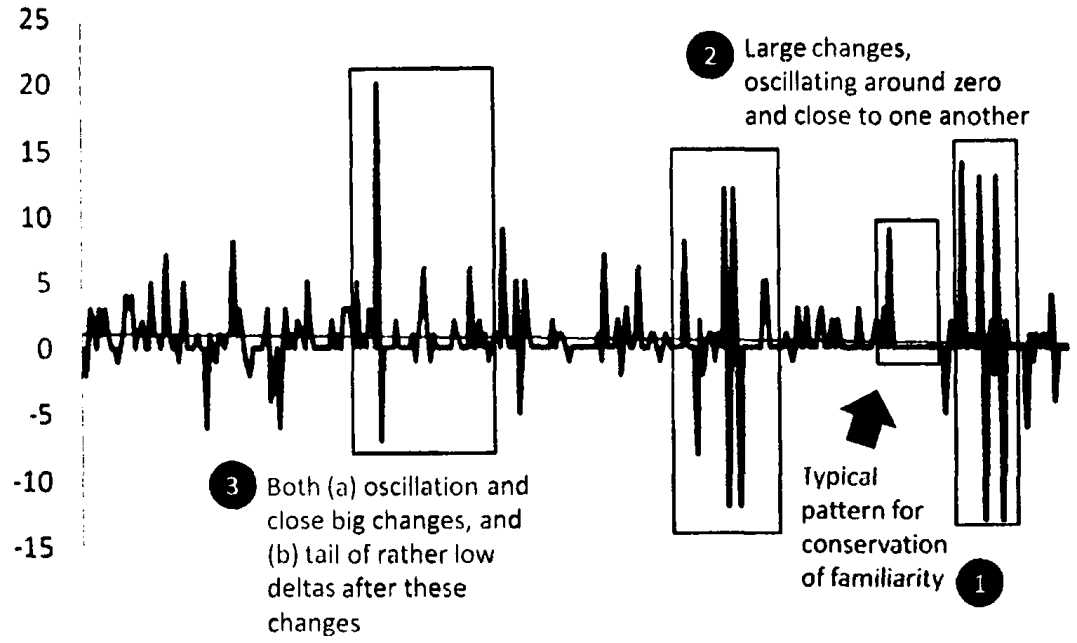


Figure 4.20 Different patterns of change in the heartbeat of MediaWiki

A second reason to question the importance of conservation of familiarity is the existence of several patterns around large changes in the growth diagram of the studied databases. Observe Figure 4.20, depicting attribute growth for the MediaWiki dataset. 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 ❶). However, it is quite frequent to see sequences of large oscillations one after the other, and quite frequently being performed around zero too (case ❷). In some occurrences, we see both (case ❸).

Thus, we *believe that the law is possible but not confirmed*; at the same time, we deem that the following wording of the law is appropriate:

Age results in a reduction of the density of changes to the database schema

---

Clearly, more work has to be done to better understand and justify the patterns of growth that are evident.

4.4.3. Discussion of our findings for the fourth law of *conservation of organizational stability.*

-To validate the hypothesis that the law of conservation of organizational stability holds, we need to establish that the project's lifetime is divided in 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).

If we stick to exact wording of the law, we can safely say that it does not hold. The multitude of spikes returning to zero after any change is such that it is impossible to speak about constant growth, even in phases. This concerns both tables and attributes.

Change is small:

- In terms of tables, growth is mostly bounded in small values. This is not directly obvious in the charts, because its shows 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, 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.

Observe Figure 4.21. Figure 4.21 has 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. In the first case, 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. The role of the lower figure is to indicate a pattern observed in all datasets: it appears that there is a Zipfian model in the distribution of frequencies.

In fact, both phenomena observed here, i.e., the bounded small change around zero, following a Zipfian distribution of frequencies is one of the few patterns that is global to all datasets, without exceptions whatsoever.

Overall, *we believe that the law does not hold*; moreover, we can replace the wording of the law as follows:

In the case of schema evolution, change follows spikes oscillating between zero and non-zero values, which are typically strongly biased towards small values close to zero.

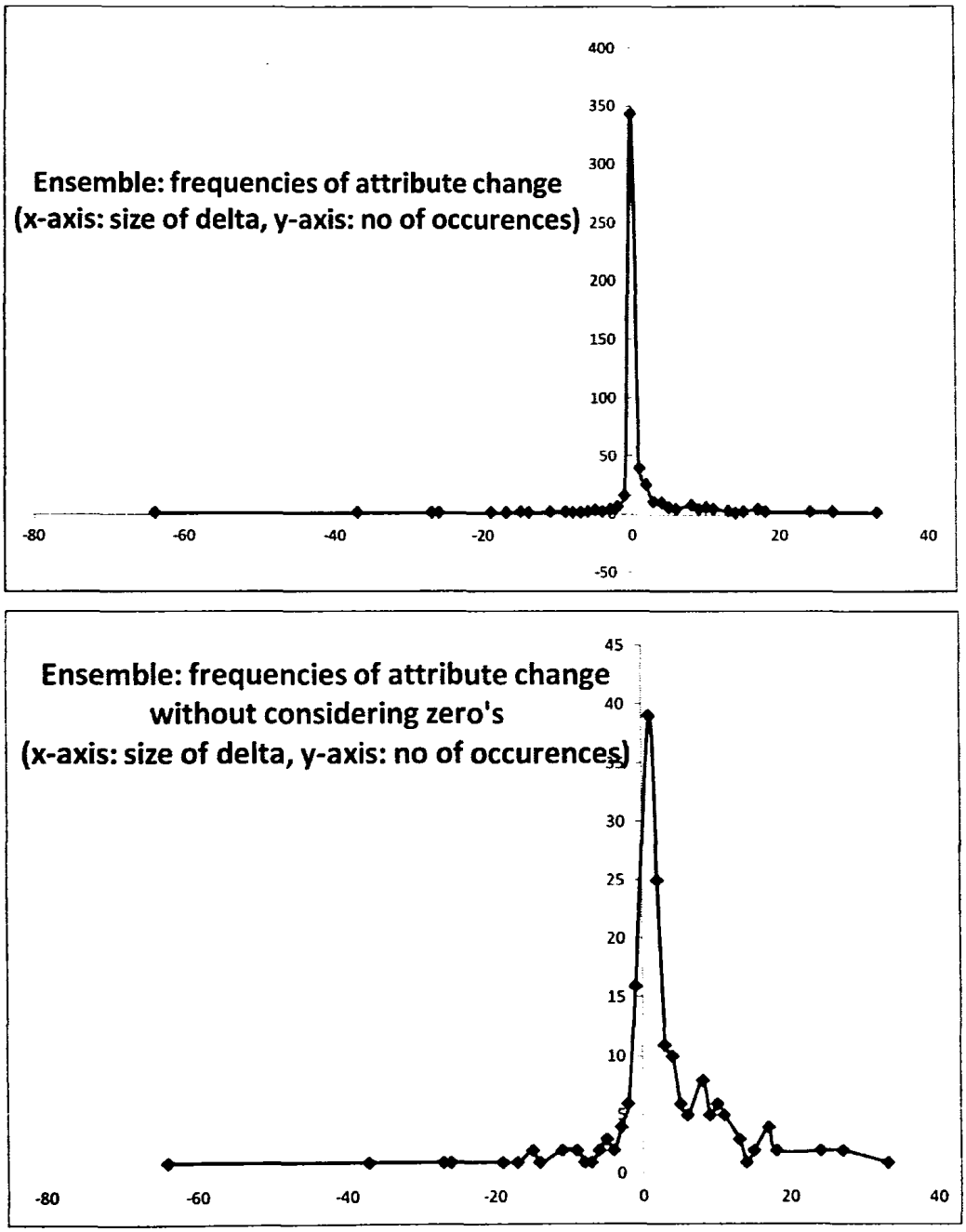


Figure 4.21 Frequency of growth size for Ensemble

4.5. Perfective Maintenance for Schema Evolution

4.5.1. Discussion of our findings for the second law of increasing complexity.

To validate the hypothesis that the law of increasing complexity holds, we need to establish the following observations: (a) there is a perfective maintenance activity that attempts to reduce complexity (demonstrated by drops in the schema size and rate of expansion), (b) similar to software systems, complexity rises (with complexity being approximated by the number of modules handled divided by growth per transition) and (c) the perfective maintenance is the negative feedback in a feedback mechanism that guides the schema evolution process.

If we take these criteria into consideration, *we see that the feedback-related criteria hold.* We frequently observe negative growth in the size of the schema, both in terms of attributes and relations. Any kind of schema shrinking is part of perfective maintenance (either because dead parts are dropped, or because redundant parts are merged, etc). At the same time, as depicted in Figure 4.7, we further observe that the evolution of the schema in terms of its actual size is in line with Lehman's regressive formula. As detailed in the discussion of the eighth law of Lehman, this strongly suggests the existence of a feedback system, which accounts for negative feedback which leads to perfective maintenance actions that aim at decreasing the complexity.

Much to our surprise, we have detected a sharp contrast between our observations and the current intuition around complexity and its evolution over time. In all datasets, except for phpBB, complexity is either stable or declines!

The above is corroborated also by the declining density of changes over time (law V): as the database ages, and as we speculate here, complexity declines, maintenance becomes easier and results in less maintenance activity.

How confident can we be on our observation? For one, we have to state that although the approximate assessment of complexity as modules handled over growth follows the literature, we cannot be certain that it accurately evaluates the complexity of a database design in the

context of its evolution. It is however, quite reassuring that this formula, is also backed up by a more concrete assessment of maintenance rate and frequency of change. The combination of the above gives a strong indication (although not certainty) for our statement on the declining complexity of database schemata.

Therefore, based on these observations we have indications that the second law holds for the examined database schemata, however with completely different connotations than the ones reported by Lehman for typical software systems: complexity drops, probably as a result of too much and too successful perfective maintenance over schemata that evolve with a relatively small evolution rate anyway.

4.5.2. Discussion of our findings for the seventh law of declining quality

The seventh law postulates that quality declines with age unless the system is rigorously adapted to its external environment. As already mentioned, the seventh law is typically supported by logical induction, as the exact measurement of quality with certainty guarantees as to what exactly we measure is not easy (if not practically impossible). So, even Lehman [LeRa06, LeRP98] suffice at making a logical proof for the law. In our case, we have assumed that the seventh law can be validated over the existence of a feedback-based system and the validity of the second law.

We are really hesitant to declare the law as valid: the feedback-system seems to exist (albeit with different characteristics than the rest of the software systems) and the second law seems to offer a case for supporting the seventh law, but only because we observe an improvement in internal quality (!). Therefore, being unsure on the internal quality already, we are even more reluctant towards declaring external quality too as improving. We have some testimonies on this: the density of schema alterations typically decreases with time (law V) which can be an indication that user requirements are gradually satisfied more and more. However, this is only an indication and can only serve as a starting point for future research.

4.6. Treats 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.

4.6.1. Construct Validity

To assess construct validity, we will review the metrics used for each law and state our concerns about their analogy with the metrics used in the studies of software evolution.

I. Continuing change

The metrics involved in this law are the changes between two schema versions over time and version ID. The information that we have in the changes of the schema is accurate and the usage of the 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 and 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 have spent in the development of the database. Moreover, the fact that databases are often found as part of other software, such a measure would not be that useful because we don't have a way to differentiate the work done on the database and the rest of the software system. On the other hand, the reasoning behind the formula used makes sense and it is consistent with the bibliography. 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. 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 the growth measure. The metric itself can be accurately measured as the incremental difference between two versions. 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 effort. As we previously mentioned the effort cannot be easily measured because effort does not always equate progress. To this end, we use the schema growth, which is accurate as we indicated before, and the changes over version which is accurately measured. Overall, we are satisfied with the approximation of effort, 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 effort.

V. Conservation of familiarity

The metrics used for the assessment of this law is growth and growth rate, both accurately measured. On the other hand, we have no way to undisputedly know the exact mechanics behind the observations. Therefore, although certain for our observations, the mechanics of the law require further elaboration.

VI. Continuing growth

For this law, we employed the number of relations and attributes for each version. Those measures were accurately extracted by our tools and are exactly fit for assessing the law.

VII. Declining quality

All measures available for the assessment of the validity of this law are good approximations. To a large extent, this is also due to the abstract wording of the law. A possible measure might be the number of defects associated with each version that we could extract via the bug-

tracking system of the software if such a system exists. Again, we emphasize the difficulty to differentiate a bug related to the database from a bug affecting the rest of the software.

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 data sets are surprisingly consistent. Therefore, we believe 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.

4.6.2. 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 have the following cases of observations, where one might be tempted to introduce a cause-effect relationship:

- Density of changes drops with age
- Complexity drops with age
- Size grows with age (in the long run)

We are very careful to treat our observations only as such and avoid relating the cause of the observed phenomena with age. In fact, due to the consistence of the observed phenomena throughout all the studied datasets, we believe that a cause-effect relationship does exist, albeit hidden, and not directly relating age with the observations. We attribute the behavior of density and complexity to the existence of a confounding variable: schema quality, which we anticipate to be closely related to age and causes the observed behavior. Still, this remains to be proved with undisputed data. 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 undisputedly by dedicated studies.

4.6.3. External Validity

This study comes within a well defined context, practically based on two pillars: (a) we restrained our study only to the core of the logical schema of the involved databases without accounting physical properties, views and constraints and (b) we only targeted databases in open source software systems.

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. The schemas collected had an adequate number of versions from rather few (40) to quite many (500+) and our findings are consistent in practically all of them. 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.

4.7. Putting it All Together

Now we are ready to revisit the original hypotheses around the mechanics of schema evolution for databases and discuss their validity and particularities.

A. Hypothesis of the feedback-based process. Is the process of schema behaving like a feed-back based system?

We believe that we can indeed claim that the overall process is guided by a feedback based mechanism, albeit with a strong negative feedback part. On the positive side of a feedback-based system, there is indeed need to incorporate more and more data in 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 also have very strong indications that the law of inverse square holds, as it applies to all eight schemata under the same two alternative formulae. However, we do not come with a good explanation as to why this holds.

B. Properties of growth

The size of the schema expands over time, albeit with versions of perfective maintenance due to the negative feedback. The expansion is mainly characterized by three kinds of phases, including abrupt change (positive and negative), smooth growth and stability (meaning large periods of no change, or very small changes).

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.

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. 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.

Although we do not believe conservation of familiarity to be a main motive, we see that the feedback mechanism of the evolution demonstrates *a reduction in the density of changes as the schema ages. This also affects the frequency of non-zero values in the spikes.* We also observe not common patterns of changes with sequences of high spikes, sometimes oscillating around zero. 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.

C. 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 has been demonstrated in several of our studies for the different laws, and is mainly demonstrated by the drops in the schema size as well as the drops in activity rate and growth with age. 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, we are negative to adopt the corroborating observations around software systems that accompany the two laws of negative feedback (II and VII) of Lehman, despite the adoption of the hypothesis for a feedback-based mechanism. We know that the measurement of complexity is an approximation; moreover, in sharp contrast to typical software systems, 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 database 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'. Apparently, 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.

Table 4.3 A summary of the results of our study.

	Original law as of 2006	Holds for Database Schemata	Our findings
I	An E-type system must be continually adapted or else it becomes progressively less satisfactory in use.	Partially holds	The change is present but not continuous.
II	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.	Seems to hold	Our approximation of complexity drops with age possibly as a result of successful maintenance.
III	Global E-type system evolution is feedback regulated.	The essence holds	Change is small.
IV	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.	Does not hold	Growth is not constant but comes with spikes, oscillating around zero with age slightly higher than zero.
V	In general, the incremental growth (growth ratio trend) of E-type systems is constrained by the need to maintain familiarity.	Possible but not confirmed	Age reduces the change frequency in a database schema
VI	The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime.	Holds	Database schema grows with age in the long run.
VII	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.	Unknown	We lack actual measures of external quality.
VIII	E-type evolution processes are multi-level, multi-loop, multi-agent feedback systems.	Seems to hold	The regression formula holds; still feedback comes with a short memory and its mechanism is not clear.

CHAPTER 5. DISCUSSION

In this final chapter, we will start with a summary of our findings. Then, we proceed discuss issues of future work.

5.1. Summary of our findings

In this subsection, we discuss fundamental *observations, conjectures* and *patterns* that have been detected in our study. We intentionally avoid the term *law*, as we do not have unshakeable evidence for their existence, from any of the two possible ways to obtain it, i.e., (a) the *empirical grounding*, due a very large amount of datasets that obey them, or (b) the *rationalized grounding*, that can be obtained via a clear explanation of the underlying mechanism that guides them, established on non-disputed facts. In fact, we could argue that we need both pillars before we establish the term 'law'. Having said that, believe that we can divide our findings in (a) evidence that we deem coming with high degree of certainty, and, (b) evidence that requires further exploration by future studies. We organize our following discussion in these two categories. We also annotate each of our observations with reference to the law where we discuss it in detail.

Before proceeding, we remind the reader 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*. We avoid generalizing our findings on 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). In all our subsequent deliberations, we take the above context as granted and avoid repeating it for reasons of better presentation of our results.

5.1.1. Observations coming with high degree of certainty

Conjecture of Feedback-based Behavior for Schema Evolution

Schema evolution demonstrates the behavior of a feedback-regulated system, as it obeys the antagonism between the need for expanding its information capacity to address user needs and the need to control the unordered expansion, with perfective maintenance.

Supporting observations:

- *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 relations 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)*

Observations concerning the heartbeat of change

- *The database is not continuously adapted, but rather, alterations occur from time to time, both in terms of versions and in terms of time. (I)*
- *Age results in a reduction of the density of changes to the database schema in most cases. (V)*

Observations concerning the size of the schema

- *As an overall trend, the information capacity of the database schema is enhanced – i.e., the size grows in the long term (VI)*
- *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)*

Schema growth is small (observations)

- *Growth is small in the evolution of database schemata; compared to typical 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 data sets. 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 – i.e., change is typically small. (IV)*
- *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)*

5.1.2. Observations requiring further investigation

Conjecture on the existence of change patterns

- *Change frequently follows a spike pattern (small change followed by no change) (III)*
- *Change comes with patterns of (a) stillness (large sequences of versions with small or zero change), (b) abrupt change (positive or negative) and (c) smooth growth (VI)*
- *Frequently, we observe patterns of large changes closely sequenced one after another (III)*

We would like to stress here that these patterns concerning growth require extensive study and verification over a large number of datasets before adopted as undisputed patterns.

Conjecture of Success of Perfective Maintenance

- *Age results in a reduction of the complexity to the database schema (II)*

We take extra care to note that the above conjecture comes with a certain degree of uncertainty around the efficacy of the approximation measure used to assess complexity.

5.2. Open issues for future work

Our results constitute a first step towards understanding the mechanics of schema evolution in databases. In the sequel, we give an indicative list of issues to be explored in some depth.

In the context of the fourth law, where the desideratum is to demonstrate that effort is constant throughout phases, we could approximate the devoted effort via measures related to time. An interesting thought, based on the original observation of the law that the effort is constant throughout the entire lifetime would be to introduce a measure of *changes per day*, which refers to the average of changes per day for the entire lifetime of the database schema (practically, we assess the amount of changes within two versions and we divide it by the time distance between them – originally, this should be constant in all time points of the system's lifetime). However, since we have several occasions where effort is highly dense in several periods (sometimes with several commits in the same day) and loose in others (sometimes with several years of stillness), we know a-priori that this does not hold. More sophisticated measures have to be thought, however, to dig out the patterns of change over time.

Of course, another open issue is how to define what an abrupt change is and how to detect the phases that the law mentions. The splitting of a lifetime in phases can allow the measurement or computing running averages over fixed number of versions and facilitate the better understanding of other laws, like for example, the fifth law (where we need to show that growth drops).

We need to do better in the *correct identification of perfective maintenance*. We do not have clear rules on what pattern of changes accurately represents the presence of perfective maintenance. We can be definitely sure in the case of large reductions in schema size, indicating the removal of dead attribute placeholders or their move in other relations; however this is not the only case. As already mentioned in this study, we work only with the exact changes that can be automatically determined; thus several actions of perfective maintenance, like attribute renaming are not captured by our method. Unfortunately, an automated method like our own can only speculate on this problem. Still, even with a (small) degree of uncertainty, such improvements would make the picture much clearer and would allow us to detect the amount and rate of perfective maintenance with better precision.

Concerning the issue of increasing complexity, we would like to point out that a major concern that must be addressed by the database community, in order to be able drive stronger conclusions on the topic is the establishment of *a representative set of metrics that measure the complexity of a database schema*. If we focus on the structural complexity only, we can identify a short list of candidates. As naïve approximations of complexity, we can use (i) the number of foreign keys (or the number of functional dependencies if a designer takes care to store them electronically) of the relational schema or (ii) the number of relationships of the conceptual schema, if such information is available. More advanced and sophisticated algorithms can be devoted to computing the number of schema components, with schema component defined as the set of relations that are semantically related to each other with a high degree of coupling. Unfortunately, measuring even the naïve approximations of complexity in the kind of experiments we conduct is typically impossible: ER models or functional dependencies are simply not available and, to a very large extent, the databases supporting CMS's or web sites are deployed without foreign keys in order to increase efficiency (and with the hope that the application will guarantee data integrity). Hence, a systematic study of database complexity, given the above practical constraints presents an interesting topic for future research. Finally assessing the laws of evolution over more datasets and extracting new patterns are two possible ways for strengthening our understanding of the mechanics of schema evolution in databases.

REFERENCES

[BeLe76] Laszlo A. Belady, M. M. Lehman. A Model of Large Program Development. IBM Systems Journal 15(3), pp: 225-252, 1976.

[CMDZ13] Carlo A. Curino, Hyun J. Moon, Alin Deutsch and Carlo Zaniolo. Automating the database schema evolution process. The VLDB Journal - The International Journal on Very Large Data Bases 22, no. 1 (2013): 73-98.

[CMTZ08] Carlo A. Curino, Hyun J. Moon, Letizia Tanca, Carlo Zaniolo. Schema evolution in Wikipedia: toward a web information system benchmark. In International Conference on Enterprise Information Systems (ICEIS 2008)

[CuMZ08] Carlo A. Curino, Hyun J. Moon, Carlo Zaniolo. Graceful database schema evolution: the PRISM workbench. Proceedings of the VLDB Endowment 1.1 (2008): p. 761-772.

[FePf96] Norman E. Fenton, Shari Lawrence Pfleeger: Software metrics - a practical and rigorous approach. International Thomson 1996, ISBN 978-1-85032-275-7.

[Leh+97] Meir M. Lehman, Juan F. Ramil, Paul Wernick, Dewayne E. Perry, Wladyslaw M. Turski. Metrics and Laws of Software Evolution - The Nineties View. 4th IEEE International Software Metrics Symposium (METRICS 1997), November 5-7, 1997, Albuquerque, NM, USA, pp: 20.

[Lehm96] M. M. Lehman. Laws of Software Evolution Revisited. 5th European Workshop on Software Process Technology (EWSPT '96), Nancy, France, October 9-11, 1996, pp: 108-124.

[LeRa01] Meir M. Lehman and J F Ramil. Software Evolution, STRL Annual Distinguished Lecture, De Montfort Univ., Leicester, 20 Dec. 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

[LeRa06] Meir M. Lehman and Juan C. Fernandez-Ramil. Rules and Tools for Software Evolution Planning and Management. In: Software Evolution and Feedback: Theory and Practice. Edited by Nazim H. Madhavji, Juan C. Fernandez-Ramil, and Dewayne E. Perry, John Wiley & Sons Ltd, 2006. ISBN-13: 978-0-470-87180-5.

[LeRP98] Meir M. Lehman, Juan F. Ramil, Dewayne E. Perry. On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution. 5th IEEE International Software Metrics Symposium (METRICS 1998), March 20-21, 1998, Bethesda, Maryland, USA, pp: 84-88.

[Press00] Roger Pressman. Software Engineering: A Practitioner's Approach: European Adaption. McGraw-Hill, 5th edition, April 2000

[PVSV12] George Papastefanatos, Panos Vassiliadis, Alkis Simitsis, and Yannis Vassiliou. "Metrics for the prediction of evolution impact in ETL ecosystems: A case study." Journal on Data Semantics 1, no. 2 (2012): 75-97.

[RaLe00] Juan F. Ramil, M. M. Lehman. Effort estimation from change records of evolving software (poster). Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000) Limerick Ireland, June 4-11, 2000, pp: 777

[Sjøb91] Dag Sjøberg, The Thesaurus – A Tool for Meta Data Management, Technical Report FIDE/91/6, ESPRIT Basic Research Action, Project Number 3070---FIDE, February 1991.

[Sjøb93] Dag Sjøberg. "Quantifying schema evolution." Information and Software Technology 35.1 (1993): 35-44.

[Skou10] Ioannis Skoulis, Hecate: SQL schema diff viewer. Bachelor Thesis, Department of Computer Science, University of Ioannina, 2010

[SEBK] IEEE. Software Engineering Body of Knowledge. IEEE – 2012 SWEBOK Guide V3 – Alpha Version. Available at http://www.computer.org/portal/web/swebok Retrieved at 13 September 2013.

SHORT VITA

Ioannis Skoulis was born in Thessaloniki in 1985 and moved to Volos at the age of five He received his BSc degree from the Computer Science Department of University of Ioannina at 2010 had his Military Service right after that. In 2011 he became an MSc student in the same institution under the supervision of Panos Vassiliadis. As a member of the DAINTINESS group, his academic interests include Software Engineering and Relational Database Evolution.