Information Systems 53 (2015) 363–385

Contents lists available at ScienceDirect

Information Systems

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

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

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

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

ARTICLE INFO

Available online 30 April 2015

Keywords:
Schema evolution
Software evolution
Lehman's laws

ABSTRACT

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

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

1. Introduction

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

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

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

http://dx.doi.org/10.1016/j.is.2015.03.009
0306-4379/

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

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

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

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

2. Lehman laws of software evolution in a nutshell

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

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

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

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

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

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

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

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