Replicability and Reproducibility of a Schema Evolution Study in Embedded Databases

Dimitri Brainingер[1], Wolfgang Mauerer[1,2], and Stefanie Scherzinger[3]

[1] Technical University of Applied Sciences Regensburg, Regensburg, Germany
dimitri.brainingер@st.oth-regensburg.de
[2] Siemens AG, Corporate Research, Munich wolfgang.mauerer@othr.de
[3] University of Passau, Germany stefanie.scherzinger@uni-passau.de

Abstract. Ascertaining the feasibility of independent falsification or repetition of published results is vital to the scientific process, and replication or reproduction experiments are routinely performed in many disciplines. Unfortunately, such studies are only scarcely available in database research, with few papers dedicated to re-evaluating published results. In this paper, we conduct a case study on replicating and reproducing a study on schema evolution in embedded databases. We can exactly repeat the outcome for one out of four database applications studied, and come close in two further cases. By reporting results, efforts, and obstacles encountered, we hope to increase appreciation for the substantial efforts required to ensure reproducibility. By discussing minutiae details required to ascertain reproducible work, we argue that such important, but often ignored aspects of scientific work should receive more credit in the evaluation of future research.

Keywords: Schema Evolution · Replicability · Reproducibility.

1 Introduction

Experiments are at the heart of the scientific process. According to the ACM reproducibility guidelines (see "ACM review and badging", hyperlink available in the PDF), experiments are expected to be *repeatable*: Essentially, the same team with the same experimental setup can reliably achieve identical results in subsequent trials. Moreover, experiments should be *replicable*, so that using the same experimental setup operated by a different team achieves the same results. Ideally, experiments are even *reproducible*, and a different team with a different experimental setup can confirm the results.

Such properties are acknowledged to be fundamental, but reproducibility is far from universally permeating most published research. This discrepancy has become an academic topic of debate, and dedicated research evaluates the (oftentimes wanting) state of affairs in computer science research in general (see, e.g., Refs. [1,5,9,12]), but also in data management research[4].

[4] Such as in the VLDB ("pVLDB Reproducibility") and SIGMOD communities ("ACM SIGMOD 2019 Reproducibility", clickable links available in PDF).

In this paper, we examine the state of replicability, and efforts required to achieve reproducibility, for an empirical case study on schema evolution in embedded databases by S. Wu and I. Neamtiu [16] that predates the aforementioned discussions. There is a long-standing tradition of schema evolution case studies in real-world database applications, e.g., [7, 13–15]. It used to be difficult to get access to real-world database applications for study, so earlier studies are generally conducted on closed-source systems, for instance [14]. Yet the proliferation of open source software, and the access to code repositories (e.g., GitHub) enables a whole new line of research on open source application code [4]. Most schema evolution studies focus on applications backed by relational database management systems, typically tracking the growth of the schema (counting the number of tables and their columns), and the distribution of *schema modification operations* (a term coined by Curino et al. in [6]).

The authors in the original case study are the first to focus on an important subfamily of database products, namely that of *embedded* (and therefore serverless) databases, such as SQLite. While there are independent schema evolution studies targeting the same open source projects, such as MediaWiki (the software powering Wikipedia), they consider different time frames (such as 4.5 years in [7] and 10 years in [13]), and implement different methodologies. This even leads to partly contradictory results. However, a dedicated replicability and reproducibility study has not yet been conducted so far.

Contributions. This paper makes the following contributions:

- We conduct a replicability and reproducibility study on a well-received, published paper on schema evolution [16]. While there is a long history of schema evolution case studies, to the best of our knowledge, ours is the first effort to ascertain published results on this class of publications.
- Our study is mainly based on the information provided in the original paper. However, we were also provided (incomplete) code artefacts by the authors of the original study. This blurs the line between conducting a replicability and reproducibility study. For simplification, we restrict ourselves to the term *reproducibility* in the remainder of this paper.
- We carefully re-engineer the authors' experiments and present our results. Overall, we achieve a high degree of accordance, albeit at the expense of substantial manual effort. For one out of four applications studied in [16], we even obtain identical results. We document and discuss where our numbers agree, and where they deviate.
- We lay out which instructions were helpful, and which left too much leeway.
- We discuss the threats to the validity of our results (e.g., where we may have erred), and contrast this with the original threats stated in [16]. Doing so, we re-calibrate the level of risk involved with each originally reported threat.

Our experience underlines that achieving full reproducibility remains a challenge even with well-designed, well-documented studies, and requires considerable extra effort. We feel that such efforts are not yet universally appreciated, albeit it is in our joint interest that research become reproducible.

```
res = logged_sqlite3_exec(sql, "CREATE TABLE file_deltas\n"
 "\t(\n"
 "\tid not null,    -- strong hash of file contents\n"
 "\tbase not null,  -- joins with files.id or file_deltas.id\n"
 "\tdelta not null, -- compressed [...]\n"
 "\tunique(id, base)\n"
 "\t)", NULL, NULL, errmsg);
```

(a) Excerpt from the C++ code in *Monotone*.

```
CREATE TABLE file_deltas
(
 id integer not null,
 base integer not null,
 delta integer not null,
 unique(id, base)
);
```

(b) Extracted stmt.

Fig. 1. (a) A CREATE TABLE statement, embedded as string constants within *Monotone* C++ code (source can be inspected online, "`[...]`" denotes a shortened comment). The statement must be automatically parsed and translated to the MySQL dialect (b).

Structure. The remainder of this paper is organized as follows. We next summarize the original study. Section 3 states our methodology. Section 4 describes the main part of the reproduction work, as well as the detailed results. Section 5 discusses the overall results, followed by Section 6 with a description of threats to validity. Finally, Section 7 focuses on related work. Section 8 concludes.

2 Original Study

We briefly summarize the original study. Neamtiu et al. analyze four database applications, all of which are based on SQLite, and provide public development histories by virtue of being available as open source software (OSS): *BiblioteQ*, *Monotone*, *Mozilla Firefox*, and *Vienna*:

- *BiblioteQ* (C++), analyzed in the time frame 03/15/2008–02/19/2010, is a library management system.
- *Monotone* (C++), analyzed in the time frame 04/06/2003–06/13/2010, is a distributed version control system.
- *Mozilla Firefox* (C, C++), analyzed in the time frame 10/02/2004–11/21/2008, is a popular web browser.
- *Vienna* (Objective-C), analyzed in the time frame 06/29/2005–09/03/2010, is an RSS newsreader for MacOS.

The original study uses a custom data processing pipeline for retrieving the source code history, extracting schema declarations embedded in application code, and computing differences between schema revisions. Extracting schema declarations requires careful engineering: Figure 1(a) shows a CREATE TABLE statement embedded in the program code as a multi-line string constant.

We compare different schema versions with `mysqldiff` (version 0.30), a utility to derive schema modification operations (SMOs) that transform a predecessor schema into the successor schema. `mysqldiff` only handles MySQL schema declarations, but SQLite uses a custom SQL dialect[5]. For instance, let us again consider the code example from Figure 1(a). The extracted CREATE TABLE

[5] The SQL dialects reference at https://en.wikibooks.org/wiki/SQL_Dialects_Reference illustrates the richness of proprietary language constructs.

Table 1. Evolution time frames and schema change details (as absolute numbers and percentages) given in the original study [16].

App	Table changes		Attribute changes				
	CREATE TABLE	DROP TABLE	ADD COLUMN	DROP COLUMN	Type change	Init change	Key change
Firefox	5 (4.2%)	26 (21.7%)	57 (47.5%)	28 (23.3%)	0 (0.0%)	3 (2.5%)	1 (0.7%)
Monotone	11 (20.4%)	17 (31.5%)	14 (25.9%)	10 (18.5%)	0 (0.0%)	0 (0.0%)	2 (3.7%)
BiblioteQ	4 (2.6%)	8 (5.2%)	27 (17.5%)	28 (18.2%)	83 (53.9%)	0 (0.0%)	4 (2.6%)
Vienna	1 (7.1%)	0 (0.0%)	13 (92.9%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
Total	21 (6.1%)	51 (14.9%)	111 (32.5%)	66 (19.3%)	83 (24.3%)	3 (0.9%)	7 (2.0%)

statement is shown in Figure 1(b). Note that the original statement does not declare attribute types, which is permissible when using SQLite. Since MySQL requires all attributes to be typed, we add a default attribute type in preparation for processing the schemas with `mysqldiff`.

`mysqldiff` generates SMOs for creating or dropping a table, adding or removing a table column, and changing the type or initial value of a column. It also recognizes changes to the table primary key. With this sequence of SMOs, the predecessor schema can be transformed into its successor schema. Further SMOs, such as renaming a table or an attribute, cannot be reliably derived based on automated analysis alone, and would require sophisticated schema matching and mapping solutions [3].

The statistics in the study by Neamtiu et al. derive from `mysqldiff` results; Table 1 provides the number of SMOs for each project. Studies on schema evolution in server-based (non-embedded) DBMS, especially [13], show that attribute type changes are frequent in many projects. In the study by Neamtiu et al., this holds only for *BiblioteQ*, so no type changes were recorded for the other projects. This is a finding that we will revisit at a later point. The original study finds that the shares of CREATE TABLE and ADD COLUMN SMOs are comparable to the observations of related studies on schema evolution in non-embedded DBMS. The observation that changes to initial values and primary keys are uncommon has also been observed in the later study of Qiu et al. [13].

3 Methodology of this Study

We conducted our reproducibility study as follows. Our code, as well as material made available to us by the original authors, is available on Zenodo (doi.org/10.5281/zenodo.4012776) to ascertain long-term availability. In particular, we publish all interim results computed by our analysis scripts (such as the extracted schemas and the results of schema comparison), for transparency.

We started with identifying the source code repositories for the four database applications, based on the information given in the original paper. Like in the original work, we wrote a script to extract the database schema declarations embedded in the source code. For *Vienna*, the authors provided us with a partial script that could not be directly made to work (caused by minor syntactic

issues, and some missing components), and was therefore re-implemented by us in Python. For all other projects, we had no such templates.

The original study used `mysqldiff` version 0.30 to compare successive schema declarations. However, we used the newer version 0.60, since the output is more succinct and also more convenient to parse. A further reason for abandoning the legacy version is that it sometimes recognizes redundant schema modification operations (as we also discuss in Section 6).

Further, the pairwise comparison of schema versions using `mysqldiff` is not very robust: A table declaration that is missing in one version (e.g., due to a parsing problem), and then re-appears later, is recognized as first dropping and later re-creating this table. This problem was pointed out in the original study, and will also be revisited in Section 6.

As a summarizing metric, we compute the difference in percentage across all SMOs observed as

$$\frac{\sum_{SMO\ s} |p(s) - r(s)|}{P},$$

where $p(s)$ is the number of changes for SMO s reported in the original publication and $r(s)$ is the number of changes for SMO s identified in our reproducibility study. Further, P is the total number of changes in the project reported in [16].

4 Results

Vienna. For *Vienna*, the authors made their raw input data available to us, so we could apply our script on the exact same data, with the exact same results.

We further attempted to locate the raw input data ourselves, based only on information provided in the original study. Unfortunately, the original Sourceforge repository no longer exists, the project is now hosted on GitHub. From there, we obtained fewer files than expected. Thus, searching for the raw input data based on the information in the paper alone would have led to a different baseline, yet the analysis still yields the same results as listed in Table 1.

Monotone. For *Monotone*, the original paper states that the study was conducted on 48 archives available from the project website. However, we have reason to believe that only 41 versions were chosen (specifically, versions 0.1, 0.2, and also from 0.10 up to and including 0.48), based on the list of available archives, as well as comments within the material that we obtained from the authors.

Moreover, it is not exactly clear from which files to extract schema declarations: In the initial versions of *Monotone*, database schemas are only declared in files with suffix `.sql`. Later, database schemas are also embedded within C++ files (starting with version 10). We therefore explored two approaches, where we (1) consider *only* schemas declared in `.sql`-suffixed files, and (2) also consider schemas embedded within the program code.

Figure 2 visualizes the results for both approaches. For each type of SMO analyzed, we compare the number of changes reported in the original study with the number of changes determined by us. Overall, our results come close. As

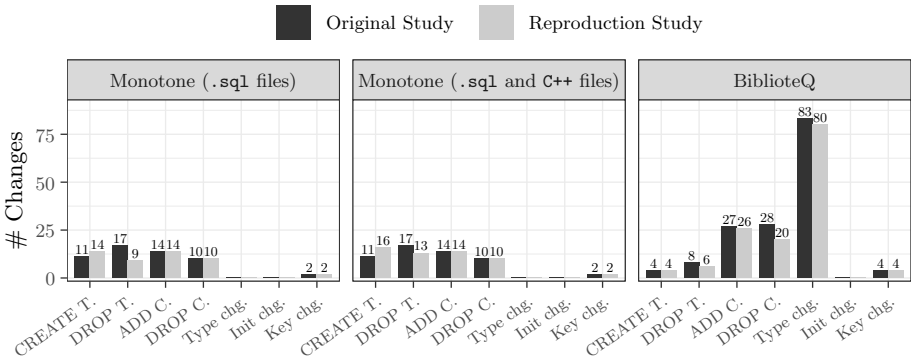

Fig. 2. Comparing the number of schema changes for *Monotone* and *BiblioteQ*.

pointed out in Section 3, problems in parsing SQL statements embedded in program code lead to falsely recognizing tables as dropped and later re-introduced. We suspect that this effect causes the discrepancies observed for CREATE and DROP TABLE statements.

BiblioteQ. At the time when the original study was performed on *BiblioteQ*, all schema declarations were contained in files with suffix `.sql` (this has meanwhile changed). Schema declarations do thus not have to be laboriously parsed from strings embedded in the application source code. MySQL, SQLite, and PostgreSQL were supported as alternative backends. In particular, SQLite was initially not supported, but was introduced with revision 35, while the original study spans the time frame from the very beginning of the project (see Section 2). Unfortunately, the original study does not discuss this issue.

We suspect that up to revision 35, the schema declarations of MySQL were analyzed, and only from then on for SQLite.[6] The high number of type changes reported for *BiblioteQ* may thus be overemphasized—the switch causes half the reported type changes. However, this still leaves a significant number of type changes for BiblioteQ, compared to the other projects (see Table 1).

The results of our reproducibility study on *BiblioteQ* are visualized in Figure 2. While we are confident that we have identified the raw input data, due to liberties in the data preparation instructions, our results nevertheless deviate.

In Table 2, we list the changes per revision, comparing the results of the original study against our own. Revision 35, where SQLite was introduced, clearly stands out. In processing the extracted schemas (in particular, revisions 4, 5 and 11), we encountered small syntax errors in SQL statements, that we manually

[6] Revision 16 only changes the MySQL schema declaration, and the original study reports a schema change in this revision. A peak in schema changes is reported for revision 35 (see Table 2), as switching from MySQL to SQLite schema declarations causes `mysqldiff` to recognize type changes. Since revision 35 only adds support for SQLite, with no schema changes for MySQL or PostgreSQL, we conclude that starting with revision 35, the authors analyzed the SQLite schema.

Table 2. Pairwise comparison of schema versions, and the number of changes w.r.t. the previous version. Stating the number of changes reported in the original paper (#C, original), the number of changes identified in our reproducibility study (#C, repro), as well as the absolute difference (diff), for *BiblioteQ*.

Revision	4	5	11	16	24	35	44	52	80	81	101	102	115	116	154	233	236	285	**Total**
#C,original	1	1	1	1	20	50	5	25	1	8	12	12	1	5	1	3	1	6	154
#C,repro	1	1	1	0	20	42	5	22	1	6	12	12	1	5	1	3	1	6	140
diff	0	0	0	1	0	8	0	3	0	2	0	0	0	0	0	0	0	0	14

Table 3. Comparing of the total number of schema changes across projects.

	Vienna	**Monotone (Alt. 1: `.sql`)**	**Monotone (Alt. 2: `.sql`/C++)**	**BiblioteQ**	**Mozilla Firefox**
Original study	14	54	54	154	120
Repro. study	14	49	55	140	–
Abs. diff	0	11	9	14	–
Rel. diff [%]	0.00	20.37	16.67	9.09	–

fixed to make the analysis work. Since we can reproduce the exact results of the original study, we may safely assume that Neamtiu et al. have fixed these same errors, even though they do not report this.

Mozilla Firefox. The original paper analyzed 308 revisions of *Mozilla Firefox* in a specific time interval. From the material provided to us by the authors, we further know the table names in database schemas. Unfortunately, this information was not specific enough to identify the exact revisions analyzed. As the original version control system (CVS) has meanwhile been replaced by Mercurial, we inspected the CVS archive, the current GitHub repositories, and the Firefox release website. We searched for the CVS tags mentioned by the authors, and tried to align them with these sources. Despite independent efforts by all three authors, we were not able to reliably identify the analyzed project versions. Consequently, we are not able to report any reproducibility results.

Summary. We summarize our results in Table 3, which reads as follows. For each project, we state the number of schema changes observed in the original study and in our reproducibility study. We state the absolute difference in the results, as well as the relative difference in percent, as introduced in Section 3.

While we were able to exactly reproduce the results for *Vienna*, we were not able to conduct the analysis for *Mozilla Firefox*. For *Monotone* and *BiblioteQ*, our results deviate to varying degrees. We next discuss these effects.

5 Discussion

Access to the raw input data, sample code and instructions make project *Vienna* an almost ideal reproduction case. For the other projects, we found the data preparation instructions unspecific. For *Monotone* and *Mozilla Firefox*, we struggled (and in case of *Mozilla Firefox* even failed) to locate the raw input data. Nearly a decade after the original paper has been published in 2011, code repositories have switched hosting platforms. Therefore, *a link is not enough* to unambiguously identify the raw input data, to quote from the title of a recent reproducibility study [12]. Further, the exact revision ranges must be clearly specified, beyond (ambiguous) dates.

The ACM reproducibility badge "Artefacts Available" requires artefacts like the raw input data to be available on an archival repository, identified by a digital object identifier. Considering our own experience, it is vital to ensure long-term access to the raw input data. Various efforts (e.g. [2]) try to ensure long-term availability of OSS repositories. However, without very specific instructions on data preparation, the reproducibility of the results remains at risk.

To quantify how much our results differ, we calculate the difference in percentage across all SMOs. For a more fine grained assessment of the degree of reproducibility, we would require information on the exact SMOs identified in the original study. This motivates us to also provide the output of applying `mysqldiff` in our reproducibility study in our Zenodo repository (see Section 3).

6 Threats to Validity

We now turn evaluate threats to the validity of the original study, and comment on additional threats discovered during reproduction.

Threats of the Original Study. Three possible threats to validity are pointed out. Firstly, missing tables in the database schema could arise from using inadequate text matching patterns. We agree that their correctness affects result quality, especially if the pattern is used to extract schemas from code that in some versions or revisions have changed significantly. Inadequate patterns can cause missing tables, missing columns, and other issues.

Secondly, renamings are another possible source of errors. Following usual schema history evolution techniques, the authors consider renaming of tables and columns as a deletion followed by an addition, as implemented by `mysqldiff`. Consequently, renamings cannot be correctly recognized.

Thirdly, the choice of reference systems is considered an external threat to validity. The evolution of database schemas for applications with different characteristics might differ.

Threats of the Reproduction Study. The dominant threat to validity of the reproduction concerns behavior of `mysqldiff`:

- Different versions of `mysqldiff` produce different output, also caused by bugs. Erroneous statements may be mistaken for actual schema changes.

- Syntax errors in table declarations cause `mysqldiff` to ignore any subsequent declarations. This error propagates, since in comparing predecessor and successor schemas, `mysqldiff` will erroneously report additional SMOs, such as DROP TABLE and CREATE TABLE statements.
- Foreign key constraints require table declarations in topological order. CREATE TABLE statements extracted from several input files require careful handling because runtime errors may cause following inputs to be ignored.

`mysqldiff` relies on a MySQL installation, and the handling of table and column identifiers in MySQL can be case-sensitive. The subject projects use lowercase table and column names, so this threat does not materialize.

Finally, incorrectly selected files containing SQL statements are a threat to validity. For instance, one individual file might be used for a specific DBMS when multiple DBMS are supported. If the schemas in different files are not properly synchronized, this leads to deviations. Carefully recording exactly which files were analyzed is necessary.

7 Related Work

The authors of the original study [8,11] analyze on-the-fly relational schema evolution, as well as collateral evolution of applications and databases. Contrariwise to the object of our study [16], the former was carried out *manually*, and risks differ between manual and programmatic analysis.

From the substantial body of work on empirical schema evolution studies, Curino et al. [7] study schema evolution on MediaWiki, and consider schema size growth, lifetime of tables and columns, and per-month revision count. They analyze schema changes at macro and micro levels. Moon et al. [10] and Curino et al. [6] test the PRISM and PRIMA systems using the data set addressed in Ref. [7], as well as SMOs to describe schema evolution. Qiu et al. [13] empirically analyze the co-evolution of relational database schemas and code in ten popular database applications. They also discuss disadvantages of using `mysqldiff`.

Pawlik et al. [12] make a case for reproducibility in the data preparation process, and demonstrate the influence of (undocumented) decisions during data preprocessing on derived results. However, we are not aware of any reproducibility studies on schema evolution.

8 Conclusion and Future Work

In this paper, we perform a reproducibility study on an analysis of the evolution of embedded database schemas. For one out of four real-world database applications, we obtain the exact same results; for two, we come within approx. 20% of the reported changes, and fail to identify the raw input data in one case.

Our study, conducted nearly a decade after the original study, illustrates just how brittle online resources are. Specifically, we realize the importance of archiving the input data analyzed, since repositories can move. This not only

changes the URL, but creates further undesirable and previously unforeseeable effects, for instance that timestamps and tags no longer serve as identifiers.

We hope that sharing our insights, we can contribute to a more robust, collective science methodology in the data management research community.

Acknowledgements. We thank the authors of [16] for sharing parts of their analysis code, and their feedback on an earlier version of this report. Stefanie Scherzinger's contribution, within the scope of project *"NoSQL Schema Evolution und Big Data Migration at Scale"*, is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) — grant number 385808805.

References

1. Abadi, D., Ailamaki, A., Andersen, D., Bailis, P., et al.: The Seattle Report on Database Research. SIGMOD Rec. **48**(4) (Feb 2020)
2. Abramatic, J.F., Di Cosmo, R., Zacchiroli, S.: Building the Universal Archive of Source Code. Commun. ACM **61**(10), 2931 (Sep 2018)
3. Bellahsene, Z., Bonifati, A., Rahm, E.: Schema Matching and Mapping. Springer Publishing Company, Incorporated, 1st edn. (2011)
4. Bird, C., Menzies, T., Zimmermann, T.: The Art and Science of Analyzing Software Data. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (2015)
5. Collberg, C., Proebsting, T.A.: Repeatability in Computer Systems Research. Commun. ACM **59**(3), 6269 (Feb 2016)
6. Curino, C.A., Moon, H.J., Zaniolo, C.: Graceful Database Schema Evolution: The PRISM Workbench. vol. 1, pp. 761–772. VLDB Endowment (Aug 2008)
7. Curino, C.A., Tanca, L., Moon, H.J., Zaniolo, C.: Schema evolution in Wikipedia: Toward a Web Information System Benchmark. In: Proc. ICEIS'08 (2008)
8. Lin, D.Y., Neamtiu, I.: Collateral Evolution of Applications and Databases. In: Proc. IWPSE-Evol'09 (2009)
9. Manolescu, I., Afanasiev, L., Arion, A., Dittrich, J., et al.: The repeatability experiment of SIGMOD 2008. SIGMOD Rec. **37**(1), 39–45 (2008)
10. Moon, H.J., Curino, C.A., Deutsch, A., Hou, C.Y., Zaniolo, C.: Managing and Querying Transaction-time Databases Under Schema Evolution. vol. 1, pp. 882–895. VLDB Endowment (Aug 2008)
11. Neamtiu, I., Lin, D.Y., Uddin, R.: Safe on-the-fly relational schema evolution. Tech. rep. (2009)
12. Pawlik, M., Hütter, T., Kocher, D., Mann, W., Augsten, N.: A Link is not Enough – Reproducibility of Data. Datenbank-Spektrum **19**(2), 107–115 (Jul 2019)
13. Qiu, D., Li, B., Su, Z.: An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications. In: Proc. ESEC/FSE'13 (2013)
14. Sjøberg, D.: Quantifying schema evolution. Information & Software Technology **35**(1), 35–44 (1993)
15. Vassiliadis, P., Zarras, A.V., Skoulis, I.: How is Life for a Table in an Evolving Relational Schema? Birth, Death and Everything in Between. In: Proc. ER 2015. pp. 453–466 (2015)
16. Wu, S., Neamtiu, I.: Schema Evolution Analysis for Embedded Databases. In: Proc. ICDE Workshops'11 (2011)