How Do Developers Document Database Usages in Source Code?

Mario Linares-Vásquez, Boyang Li, Christopher Vendome, and Denys Poshyvanyk
The College of William and Mary
Email: {mlinarev, boyang, cvendome, denys}@cs.wm.edu

***Abstract*—Database-centric applications (DCAs) usually contain a large number of tables, attributes, and constraints describing the underlying data model. Understanding how database tables and attributes are used in the source code along with the constraints related to these usages is an important component of DCA maintenance. However, documenting database-related operations and their constraints in the source code is neither easy nor common in practice.**

In this paper, we present a two-fold empirical study aimed at identifying how developers document database usages at source code method level. In particular, (i) we surveyed open source developers to understand their practices on documenting database usages in source code, and (ii) we mined a large set of open source projects to measure to what extent database-related methods are commented and if these comments are updated during evolution. Although 58% of the developers claimed to find value in method comments describing database usages, our findings suggest that 77% of 33K+ methods in 3.1K+ open-source Java projects with database accesses were completely undocumented.

I. Introduction

Database-centric applications (DCAs) are software systems that rely on databases to persist records using database objects such as tables, columns, constraints, among the others. These database objects represent the underlying application model, including business rules and terms. Also, developers can create queries and views as a mechanism to traverse or search over the persisted data by following the semantics defined by the database objects. DCA architectures are commonly used for different types of systems ranging from large enterprise systems to small mobile applications. It is not uncommon for many modern DCAs to contain databases comprised of thousands of tables and attributes [1–3].

The source code and database schemas of DCAs are constantly evolving, oftentimes asynchronously [4]. This makes it particularly challenging for developers who need to understand both how the database is used in the source code and how the model is described by a schema [5].

In addition, database administrators who are in charge of database schemas may not necessarily be in charge of related source code changes [5] or be able to effectively communicate to developers the modifications to the database schemas. Therefore, complete and up-to-date documentation of the database, the schema, and any constraints is an important artifact to support software evolution. Some existing artifacts designed to capture database schemas are data dictionaries describing all the database objects in a given schema or diagrams (conceptual and physical) depicting tables, attributes, and their relationships. However, navigating and understanding such artifacts can be tedious and time consuming tasks, in particular for large databases.

Source code comments are another source of documentation that can help developers understand nuances of the data model and database usages in the source code. However, recent studies on co-evolution of comments and code showed that the comments are rarely maintained or updated, when the respective source code is changed [6, 7]. Another study by Kajko-Mattsson showed that none of the organizations for eighteen enterprise systems have fully matched all their documentation requirements [8]. To understand if and how database-related statements are commented in source code, we mined Java applications in GitHub that use JDBC for the data access layer, and we found that 77% of 33K+ source code methods do not have header comments; in the case of existing comments, they rarely got updated when related source code was modified. To complement the mining-based analysis, we conducted a survey with 147 open-source developers (Section III). As it turns out, developers rarely write comments detailing database schema constraints (e.g., unique values, non-null keys, varchar lengths) to which developers should adhere in the source code nor document changes in source code dealing with databases. However, despite the preference of developers for using database documentation or schemas when understanding data models, 65.99% of the surveyed developers consider that tracing schema constraints along call chains in source code is not a “very easy” nor “easy” task. Therefore, there is a clear opportunity for researchers to propose approaches that support automated documentation of source code involving database operations.

To the best of our knowledge, no previous work has been done to understand database documentation practices at source code level. Therefore, this paper has the following noteworthy contributions: (i) results from a survey with open source developers to understand their practices on documenting database usages in source code; (ii) a mining-based analysis of a large set of open source projects to measure to what extent database-related methods are commented and if these comments are updated during evolution.

II. State-of-the-art: Related Work

Despite recent studies showing that there is a strong evolutionary coupling between database schema and source code [4, 9–11], as of today, no approach has been proposed to automatically document/comment database usages in source code. Some approaches have been proposed to extract information directly from the schema, but without considering the

source code [3, 12–15]. There is also some previous work for automated comment generation of software artifacts [16–28]; however, none of the existing approaches focus on generating database-related documentation for supporting evolution and maintenance of source code.

A. Studies on Co-evolution of Schema and Code

Maule et al. [11] use program slicing and dataflow-based analysis to identify the impact of database schema changes. Qiu et al. [4] conducted an empirical study into co-evolution between database schemas and source code, and the authors demonstrated that database schemas frequently evolve with many different types of changes at play. Sjøberg [10] presents a technique for measuring the changes of database schemas and performed a study on health management systems over several years, where additions and deletions are found to be the most frequent operations.

B. Extracting Database Information

Several studies focus on extracting database-related information [12, 13]. The approach proposed by Petit et al. first extracts the database table names and attributes from the database schema [12]. Then, the approach builds semantic relations between the entities by investigating set operations and join operations. Alhajj et al. presented an algorithm to identify candidate and foreign keys of all relations from an existing legacy database [14].

Another group of studies focused on analyzing data in the database and extracting associative constraints [3, 15]. The associative rule mining problem was first introduced by Agrawal et al. [3], where the associative rule mining algorithm is able to generate a set of implications $A \rightarrow B$ based on the given relational table. Au et al. [15] applied a fuzzy association rule mining technique to a bank database system and identified some hidden patterns in the data. Li et al. [29] use association rule mining technique to correct the semantic errors in generated data. Overall, the associative rule mining algorithms rely on existent data records.

C. On Documenting Software Artifacts

Buse and Weimer [16] present an approach for generating human-readable documentation of exceptions in Java. More specifically, they use a method call graph and symbolic execution techniques to extract the conditions of exceptions. Then, they use a predefined template to generate natural language comments. Sridhara et al. present an approach for automatically generating summary comments for Java methods [17]. The authors demonstrate how to identify different kinds of important lines of code based on various characteristics of the code. Once these important lines are identified, the technique converts them into natural language phrases within their method bodies. Moreno et al. later extended the scope of the comment generation to class level granularity [18, 19]. Their description is based on superclass, stereotypes of the class, and behaviors of blocks. McBurney and McMillan [28] uses contextual information (i.e., most important method in the context of a target method) to generate method summaries that include natural language descriptions of how to use the method and the purpose of the method as a part of a codebase.

Differently from the previous work, a number of papers focused on summarizing differences between program versions [20–27]. Linares-Vásquez et al. implemented a tool, ChangeScribe, for automatically generating commit messages [20, 21]. They extracted changes between two adjacent versions of a project and identified involved change types in addition to performing commit level stereotype analysis. Moreno et al. [27] introduced an approach, ARENA, for automatic generation of release notes of Java systems. Jackson and Ladd [24] present a tool named SematicDiff, which uses a program analysis technique to summarize the semantic differences between two versions of a project. Canfora et al. [25] present Ldiff to find line-based differences between two versions. More specifically, Ldiff is able to track lines moved away from the original position by comparing all combinations of *diff* fragments. Buse and Weimer [26] present an approach for generating human readable documentation for program differences based on symbolic execution.

III. State-of-the-practice: An Empirical Study and A Survey

In order to understand developer practices regarding documenting database usages in source code, we designed an empirical study composed of (i) an online survey with contributors of open source Java projects at GitHub and (ii) a mining-based analysis of method comments in these projects. In particular, the *goal* of this study is to understand how developers document or comment methods in source code that invoke database queries or statements. As for the context, we analyzed 3,113 open source Java projects at GitHub (with JDBC API calls executing SQL queries/statements) and the complete change history of 264 of those projects; we also surveyed 147 developers contributing to these projects.

A. Research Questions:

Commenting database related operations and schema constraints in source code is not a common practice, because comments in source code are mostly promoted as a way to describe the purpose of a code entity (e.g., class, method). Also, there is the assumption of the existence of artifacts covering database documentation (i.e., external documentation). However, this is not always the case, because (i) external documentation can be outdated, and (ii) understanding large database schemas is a time consuming task. Also despite of the existence of database documentation, it is possible that some database models are more clear for developers when the database objects are described in the context of features implemented in the source code.

One hypothesis that we started to explore in this paper is that inferring database schema constraints from the source code is not an easy task, and less information about the database can be inferred from the source code methods at higher levels of the call-chains. Therefore, the source code methods that are closer to the user interaction (i.e., the GUI layer) are closer to high-level actions and decoupled from the internal details about the database usages and schema constraints. As an initial effort to explore our hypothesis, in this paper we aimed at answering the following research questions:

RQ$_1$ *Do developers comment methods in source code that locally execute SQL queries and statements?*

RQ$_2$ *Do developers update comments of database-related methods during the evolution of a system?*

RQ$_3$ *How difficult is it for developers to understand propagated schema constraints along call-chains?*

RQ$_1$ examines the extent that methods in which SQL queries/statements occur are commented and developers' motivation for commenting (or not) the methods. We consider both responses by developers and a mining-based analysis of source code. **RQ**$_2$ investigates whether comments in headers of methods related to database accesses are likely to be outdated or are modified prior to new releases of the DCA. In **RQ**$_2$, similarly to **RQ**$_1$, we compare the responses from developers to an analysis of the source code at release granularity. While the co-evolution of comments and code have been studied before [6, 7], our study is the first one to investigate co-evolution of database related source code and comments. **RQ**$_3$ investigates whether tracing schema constraints in source code methods that are involved in database-related call-chains is a difficult task.

B. Data Collection

In order to answer our research questions, we identified a list of 381,161 Java projects on GitHub. We used GitHub's public API [30] to extract all Java projects and locally clone them. We then applied a keyword search to all of the files in each repository with the *.java* extension. In order to identify the projects using SQL queries, we used the following keywords *import java.sql*, *jdbc:mysql*, or *DriverManager.getConnection*. This resulted in 18,828 Java projects using JDBC. We further refined this list by removing projects that were a fork and did not have at least one star or watcher (this filtering aims to avoid projects that are duplicates or abandoned projects). In the end, we had 3,113 projects in our dataset. We extracted the developers of each project and filtered their emails down to 12,887 by removing emails containing *@(none)* and *@localhost* as well as using a regular expression to enforce proper email formatting (^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+$). Due to a limitation of the survey platform, we invited 10,000 developers to participate in our survey hosted on Qualtrics [31].

The survey consisted of five questions (**Q**$_1$ - **Q**$_5$) to understand the extent to which developers document database interactions in source code and their experience with maintaining this documentation. The questions are listed in Table I. In the table, we define "top-level method callers" as methods in a public interface that start a call-chain that trigger methods with SQL queries/statements. Concerning the mapping between the research questions and the survey questions: **Q**$_1$ and **Q**$_2$ relate to **RQ**$_1$; **Q**$_3$ and **Q**$_4$ relate to **RQ**$_2$; and **Q**$_5$ relates to **RQ**$_3$;

In addition to the survey, for **RQ**$_1$, we counted the number of source code methods with header-comments in the 3,113 projects. We analyzed the latest snapshot of each project by extracting the project's abstract syntax tree (AST). Then, we automatically detected the methods with database queries/statements to extract the package, class, method name, number of parameters, and method comments when available. We focused on JDBC API calls executing SQL queries or statements. We performed the same analysis at release-level for 264 of the projects and compared the comments for each release to understand whether developers update the comments during the project's evolution, in part answering **RQ**$_2$.

TABLE I. DEVELOPER SURVEY QUESTIONS AND RESULTS.

Question/Answer	Respondents	
Q$_1$. Do you add/write documentation comments to methods in the source code? (i.e., comments in the header of the method declaration)		
Yes	122	82.99%
No	25	17.01%
Q$_2$. Do you write source code comments detailing database schema constraints (e.g., unique values, non-null keys, varchar lengths) that should be adhered by the developers in the source code?		
Yes	32	21.77%
No	115	78.23%
Q$_3$. How often do you find outdated comments in source code?		
Never	1	0.68%
Rarely	28	19.05%
Sometimes	80	54.42%
Fairly Often	35	23.81%
Always	3	2.04%
Q$_4$. When you make changes to database related methods, how often do you comment the changes (or update existing comment) in the methods, the callers, and all the methods in the call-chains that include the changed methods?		
Never	37	25.17%
Rarely	34	23.13%
Sometimes	45	30.61%
Fairly Often	14	9.52%
Always	17	11.56%
Q$_5$. How difficult is it to trace the schema constraints (e.g., foreign key violations) from the methods with SQL statements to top-level method callers?		
Very Easy	14	9.52%
Easy	36	24.49%
Moderate	66	44.90%
Hard	23	15.65%
Very Hard	8	5.44%

C. Results

RQ$_1$**:** It is worth noting that for **RQ**$_1$ we were interested in answers concerning the general usage of documentation as a practice instead of the frequency. Therefore, **Q**$_1$ and **Q**$_2$ are dichotomous questions, and the participants had the chance of complementing the answer with a open field explaining their rationale. Regarding the results, developers mostly recognize the importance of commenting source code methods as a way to increase program comprehension during evolution and maintenance. For instance, 122 developers answered "Yes" to **Q**$_1$ and augmented the response with rationale like:

> *"Helps explain to others what the method is doing and how to use it, as well as remind the original developer (me) what was the intention of the code"*

> *"Comments make it easier to remember what things do. They are super helpful when returning to old code and when sharing your code with others. I've programmed without comments in the past and learned the hard way that comments are, more often then not, indispensable"*

From the 17.01% of participants answering "No" to **Q**$_1$, we found typical rationale claiming that methods should be self-documented/self-explanatory.

Despite the high rate of participants recognizing the practice of commenting source code methods, the answers for **Q**$_2$ predominantly indicate that the developers do not comment database schema constraints with 78.23% of the respondents answering "No" to **Q**$_2$. This contrast demonstrates that database-related information is not likely contained in the method declarations, because method comments are mostly for describing the purpose of the method instead of implementation details, and the documentation of database objects is an obligation of an external document or the database schema.

Some responses from participants supporting the preference for external documentation are the following:

"The database schema and documentation takes care of that. I can always look at the table definition very easily."

"I use [database-engine] and the constraints can be checked typing a sql consult or even using a Tool."

"Although I strongly believe comments are important, database comments are the gray area. Comments related to the database schema and its constraints I consider to be irrelevant to the code using it. The schema, its details, and any quirks about it should be outlined in a separate document."

"The comments should be stored in the database itself - which is not supported by most databases I know. Writing in the source code means duplicate effort - need to keep the source code synchronized with the schema. Also, the database is sometimes accessed by programs written in different languages - maintaining the comments up-to-date in ALL the source codes would be impossible."

Another prominent reason for not documenting database usages and constraints in method headers is the usage of Java Annotations in ORM frameworks, which explicitly declare in the code the schema constraints:

"This can be mostly handled through proper design. If using an ORM we can specify field lengths in attributes, that can provide validation as well as documenting if for developers. "

"ORM initialization makes it clear what the scheme constraints are"

"The schema is already described in ORM code."

Results from the mining study confirm developers preferences. In the analyzed source code (i.e., 3,113 projects), we identified a total of 33,045 methods invoking SQL queries/statements. Of these methods, 25,450 did not have any comment, while 7,595 methods were commented. These numbers reinforce the result of $\mathbf{Q}_2$ since 23% of the methods with database access were documented and 21.77% of developers indicated that they do in fact document such database interactions.

Summary for RQ$_1$. While developers indicated that they documented methods, we found 77% of methods with database access were completely undocumented. In fact, 115 out of 147 (78.23%) surveyed developers consider that documentation of schema constraints should not be included in the source code and it is a responsibility of the schema or external documentation.

RQ$_2$: In order to understand whether developers update comments relating to database queries/statements, we sought to understand the prevalence of outdated comments ($\mathbf{Q}_3$). Combining "Never" and "Rarely," 19.73% of developers suggest that comments are regularly updated in the systems that they implement or utilize. The remaining 80.27% of developers find outdated comments. Of those 80.27% of developers, 25.85% indicated a relative high frequency of encountering outdated comments. These results suggest that outdated comments are relatively prevalent (i.e., they are not a rare occurrence), which confirms that the comments are rarely maintained or updated, when the respective source code is change [6, 7].

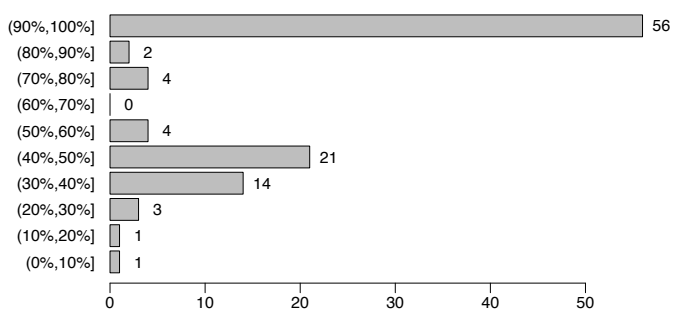

Fig. 1. Frequency of methods grouped by the ratio between the number of changes to the header comment and number of method changes in methods invoking SQL queries/statements.

When we consider the prevalence of developers updating their own comments regarding changes to database-related methods ($\mathbf{Q}_4$), we found 50.3% of respondents rarely or never updated the comments. Only 21.08% of respondents updated comments with a relatively high frequency (i.e., fairly often or always). The remaining 28.62% indicated it was neither a rare nor frequent occurrence. Therefore, the survey results suggest that it would fairly probable for such comments to be outdated. These results reinforce the problems that we observed with $\mathbf{RQ}_1$. Not only are methods with database queries undocumented, but in the case of commented methods they are also likely to be outdated.

We also analyzed $\mathbf{RQ}_2$ by relying on open source systems. We mined 264 projects that had explicit releases in GitHub to identify whether methods invoking database queries/statements updated their comments. Overall, developers did not update the comments when the methods were changed. We found 2,662 methods that invoke SQL queries/statements in the 264 projects. Of these 2,662 methods, 618 methods were updated during the history of these projects and experienced a total of 1,878 changes. 512 out of the 618 methods that changed did not have changes to their comments. The 512 method experienced on average 2.5 changes ($min = 1$, $Q_1 = 1$, $Q_2 = 2$, $Q_3 = 2$, $max = 199$) during their entire history. The rest of 106 methods (17.15%) were changed 597 times and experienced on average 5.63 changes ($min = 1$, $Q_1 = 2$, $Q_2 = 3$, $Q_3 = 5.75$, $max = 198$). In those 106 methods, we found 459 out of 597 method changes also experienced an update to the method comment. Finally, we computed the ratio between changes to header comments and methods changes; a 100% ratio means that every time a method was changed, the header comments was also changed. Figure 1 depicts the ratio of changes between header comments and source code for the 106 methods that experienced changes. For instance, we only found 66 methods in which more than 50% of the method changes were accompanied by a change to the header comment.

Summary for $\mathbf{RQ}_2$. While approximately half of the developers indicated that they "rarely" or "never" update method comments for database-related methods, we empirically observed that only 17.15% of methods that were changed in 3,113 open source projects also had their comments updated at least once between releases. Thus, we empirically found database-related methods are far less frequently commented during evolution.

$\mathbf{RQ}_3$: Answers to $\mathbf{Q}_5$ show a different perspective. Despite most of the developers thinking database documentation and schema are enough to understand schema constraints and they do not document database-related methods, answers to $\mathbf{Q}_5$ show that only 34.01% of respondents indicated that it was "easy" or "very easy" to trace database constraints along the call-chain to the top-level caller methods. The remaining 65.99% found it at least moderately difficult, with 21.09% indicating it was "hard" or "very hard." These responses indicate that tracing database constraints along call-chains is a non-trivial task. A call-chain represents the implementation of a feature in source code. This suggest that, even if external documentation or database schema is available, maintaining or implementing a new feature of a system involving database operations may be a non-trivial task, because of the effort required to trace schema constraints across the call chains. However, more empirical validation is required to support this claim.

Summary for $\mathbf{RQ}_3$. Surveyed developers prefer to rely on external database documentation and two-thirds of developers indicated tracing constraints along the call-chain was a "moderate" challenge or a "very hard" challenge. This opens the discussion about whether external database documentation is enough for supporting source code evolution and maintenance tasks.

D. Discussion

This preliminary study suggests that (i) documenting database usages and constraints is not a common practice in source code methods, (ii) developers do not update comments when changes are done to database-related methods, and (iii) tracing schema constraints through call-chains in the call graph is not an easy task in most of the cases. While results for $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$ describe developers rationale for not documenting database related operations in source code, the findings in $\mathbf{RQ}_3$ present a different perspective in terms of whether current practices for documenting databases are enough or useful for supporting developers. Schemas and database documentation have the purpose of describing the physical data model supporting a system. However, it is still unclear if this type of documentation is effective and efficient when maintaining software systems supported on large databases. Another aspect is the quality of the documentation; when the schema is complicated or hard to access, documentation is the last line of defense for understanding/querying the schema. However, there is also always the problem of outdated documentation.

In addition, understanding/querying updated documentation can be a time-consuming task, when it is not designed to support easy browsing or a specific task. For instance, let's assume a scenario in which a maintainer wants to identify the schema constraints that are involved in a feature implemented in the source code. A feature may involve different operations on several database objects, and database documentation and schemas are not intended to describe constraints and relationships at the feature-implementation level. Therefore, the maintainer has to identify the constraints by exploring the code and understanding the available documentations. Moreover, current approaches for automated documentation aim at describing the purpose of a code entity (e.g., class, method), but neither target specific evolution/maintenance tasks nor describe the entity as part of an architecture (i.e., the description of a code entity in the GUI layer should not follow the structure of a description for an entity in the data access layer).

In summary, future work should be devoted to providing developers with tools for automatic documentation that support specific tasks, in particular for evolution and maintenance of DCAs. Our results suggest that automatic generation of database-related documentation is required to support evolution/maintenance tasks. Using the results in $\mathbf{RQ}_1$ as a reference, developers working on features involving the 25,450 undocumented database-related methods — from the 3,113 analyzed projects — might find benefit in an automated approach that assures the methods are properly documented and updated. Also, the automated approach might benefit the 65.99% of the surveyed developers that did not consider tracing schema constraints along call chains as a "very easy" nor "easy" task.

E. Threats to Validity

The *construct* threat to validity relates to the observations from the developer survey and results of mining the database usage. In terms of our survey, we did not infer behavior from the survey and only reported the experience as indicated by developers and we do no provide rationale for the observations. Since we relied on projects on GitHub, it is possible that project histories are incomplete due to the relative young age of the forge or that releases were not properly tagged. In terms of *internal* threats to validity, it is possible that the developers responding to the survey had more difficulty with database-related documentation. However, the results of the survey suggest that participants had a range of experience and no single response was overly represented, which would indicate a clear bias. The *external* threats to validity relate to generalizing the results. We do not assert that the results apply to all developers or developers using other database models (e.g., ORM). Our results represent a subset of Java projects on GitHub, and other languages or forges may produce a different findings. However, GitHub is the most popular forge and our approach applies to Java projects using JDBC only.

IV. Conclusion and Future Work

In this paper, we presented a preliminary study aimed at identifying how developers document database usage at method-level in source code . The study was two fold: (i) we surveyed 147 open source developers about their practices for documenting database usages and constraints at source code level, and (ii) we mined the source code of 3,113 open source systems to measure the amount of database-related methods with header comments, and the coevolution of the comments with method changes. Our main findings suggest

that documenting methods with database accesses is not a common practice, and the the surveyed developers consider that documentation of schema constraints should not be included in the source code and it is a responsibility of the schema or external documentation. However, the impact of using database documentation to support evolution and maintenance tasks needs to be evaluated and analyzed more carefully by the research and industry communities. For instance, despite the high confidence of surveyed developers on database documentation, about 66% of them answered that tracing schema constraints along call-chains in the source code (which is a common task for maintainers of DCAs) is a moderate/very hard challenge. Therefore, future efforts on automatic approaches for artifacts documentation should also consider information extracted from the database and should be tasks-oriented instead of being general purpose. By the time this paper was written and to the best of our knowledge, no approach has been developed to generate documentation that takes into account the data model represented in database schemas, and there is no approach available that supports database schema understanding during evolution and maintenance of DCAs.

Acknowledgements. This work is supported in part by the NSF CNS-1510239, CCF-1253837, CCF-1525902 and CCF-1218129 grants. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

REFERENCES

[1] F. Rahman, D. Posnett, I. Herraiz, and P. Devanbu, "Sample size vs. bias in defect prediction," in *FSE'13*, 2013, pp. 147–157.

[2] K. Bakshi, "Considerations for big data: Architecture and approach," in *IEEE Aerospace Conference*, 2012, pp. 1–7.

[3] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD Record*, vol. 22, no. 2, 1993, pp. 207–216.

[4] D. Qiu, B. Li, and Z. Su, "An empirical analysis of the co-evolution of schema and code in database applications," in *FSE'13*, 2013, pp. 125–135.

[5] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268–1287, 1988.

[6] B. Fluri, M. Wursch, and H. Gall, "Do code and comments co-evolve? on the relation between source code and comment changes," in *WCRE'07*, 2007, pp. 70–79.

[7] B. Fluri, M. Würsch, E. Giger, and H. C. Gall, "Analyzing the co-evolution of comments and source code," *Software Quality Journal*, vol. 17, no. 4, pp. 367–394, 2009.

[8] M. Kajko-Mattsson, "A survey of documentation practice within corrective maintenance," *Empirical Software Engineering*, vol. 10, no. 1, pp. 31–55, 2005.

[9] M. Goeminne, A. Decan, and T. Mens, "Co-evolving code-related and database-related changes in a data-intensive software system," in *CSMR-WCRE'14*, 2014, pp. 353–357.

[10] D. Sjøberg, "Quantifying schema evolution," *Information and Software Technology*, vol. 35, no. 1, pp. 35–44, 1993.

[11] A. Maule, W. Emmerich, and D. S. Rosenblum, "Impact analysis of database schema changes," in *ICSE'08*, 2008, pp. 451–460.

[12] J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian, "Towards the reverse engineering of renormalized relational databases," in *International Conference on Data Engineering*, 1996, pp. 218–227.

[13] J.-M. Petit, J. Kouloumdjian, J.-F. Boulicaut, and F. Toumani, "Using queries to improve database reverse engineering," in *Entity-Relationship Approach—ER'94 Business Modelling and Re-Engineering*. Springer, 1994, pp. 369–386.

[14] R. Alhajj, "Extracting the extended entity-relationship model from a legacy relational database," *Information Systems*, vol. 28, no. 6, pp. 597–618, 2003.

[15] C. M. Kuok, A. Fu, and M. H. Wong, "Mining fuzzy association rules in databases," *ACM Sigmod Record*, vol. 27, no. 1, pp. 41–46, 1998.

[16] R. P. Buse and W. R. Weimer, "Automatic documentation inference for exceptions," in *ISSTA'08*, 2008, pp. 273–282.

[17] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," in *ASE'10*, 2010, pp. 43–52.

[18] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for java classes," in *ICPC'13*, 2013, pp. 23–32.

[19] L. Moreno, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Jsummarizer: An automatic generator of natural language summaries for java classes," in *ICPC'13*, 2013, pp. 230–232.

[20] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *SCAM'14*, 2014, pp. 275–284.

[21] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, "Changescribe: A tool for automatically generating commit messages," in *ICSE'15, Formal Research Tool Demonstration*, 2015, pp. 709–712.

[22] H. A. Nguyen, T. T. Nguyen, H. V. Nguyen, and T. N. Nguyen, "idiff: Interaction-based program differencing tool," in *ASE'15*, 2011, pp. 572–575.

[23] C. Parnin and C. Görg, "Improving change descriptions with change contexts," in *MSR'08*, 2008, pp. 51–60.

[24] D. Jackson and D. A. Ladd, "Semantic diff: A tool for summarizing the effects of modifications," in *ICSM'94*, 1994, pp. 243–252.

[25] G. Canfora, L. Cerulo, and M. Di Penta, "Ldiff: An enhanced line differencing tool," in *ICSE'09*, 2009, pp. 595–598.

[26] R. Buse and W. Weimer, "Automatically documenting program changes," in *ASE'10*, 2010, pp. 33–42.

[27] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, A. Marcus, and G. Canfora, "Automatic generation of release notes," in *FSE'14*, 2014.

[28] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *ICPC'14*, 2014, pp. 279–290.

[29] B. Li, M. Grechanik, and D. Poshyvanyk, "Sanitizing and minimizing databases for software application test outsourcing," in *ICST'14*, 2014, pp. 233–242.

[30] "GitHub API. https://developer.github.com/v3/. Last accessed: 2015/01/15."

[31] "Qualtrics. http://www.qualtrics.com."