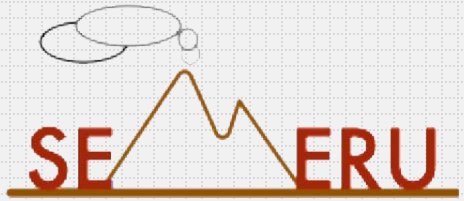
How Do Developers Document Database Usages in Source Code?

Mario Linares-Vasquez, Boyang Li, Christopher Vendome, and Denys Poshyvanyk



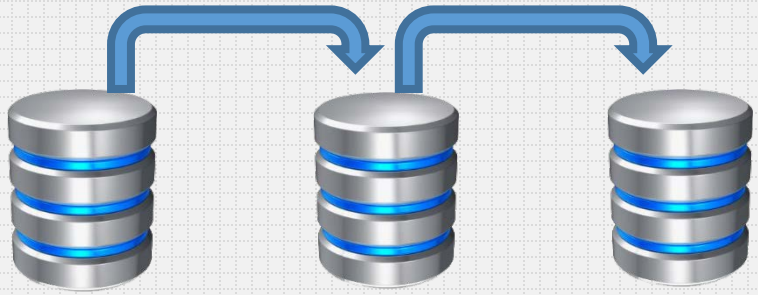
Database-centric application (DCA)

DCAs are software systems that rely on databases to persist records using database objects.

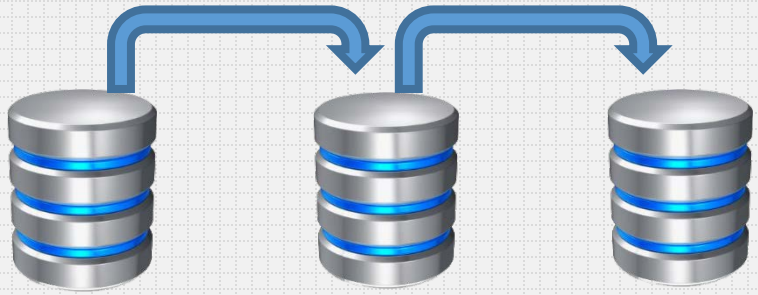
Database-centric application (DCA)

DCAs are software systems that rely on databases to persist records using database objects.

Challenges



Challenges

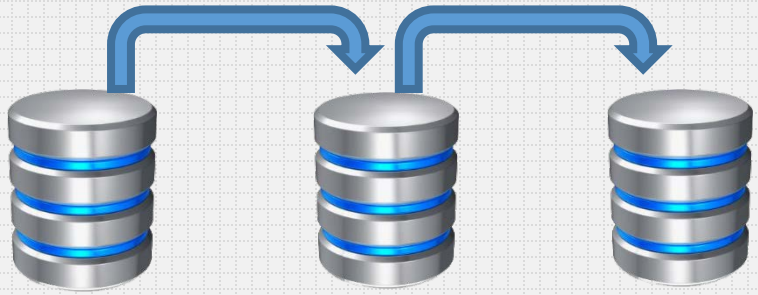


STUDENT

ID	Num	PWD	Gender	Address	Year	…
…		…	…	…	…	…
…		…	…	…	…	…

DBManager.getAllInfoByStudentID ()

Challenges



STUDENT

ID	Num	PWD	Gender	Address	Year	…
…		…	…	…	…	…
…		…	…	…	…	…

ID	Num	PWD	…
…	…		…
…	…		…

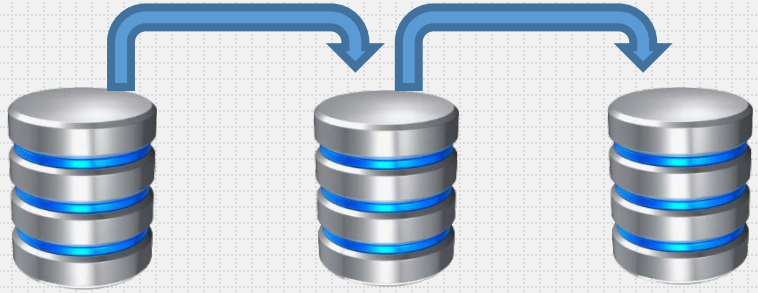
ST_LOGIN

ID	Gender	Address	Year	…
…	…	…	…	…
…	…	…	…	…

ST_DETAILS

DBManager.getAllInfoByStudentID ()

Challenges



STUDENT

ID	Num	PWD	Gender	Address	Year	…
…		…	…	…	…	…
…		…	…	…	…	…

ID	Num	PWD	…
…	…		…
…	…		…

ST_LOGIN

ID	Gender	Address	Year	…
…	…	…	…	…
…	…	…	…	…

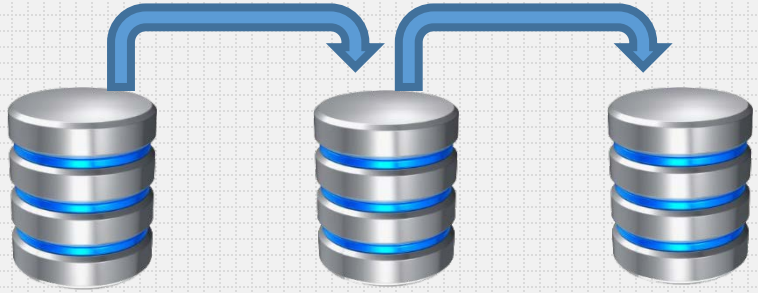
ST_DETAILS

DBManager.getAllInfoByStudentID ()

getSTLogin()

getSTDetails()

Challenges



STUDENT

ID	Num	PWD	Gender	Address	Year	…
…		…	…	…	…	…
…		…	…	…	…	…

ID	Num	PWD	…
…	…		…
…	…		…

ST_LOGIN

ID	Gender	Address	Year	…
…	…	…	…	…
…	…	…	…	…

ST_DETAILS

UI.student.buttonClickShowAllInfo()

UI.student.quaryAllInfoByID ()

DBManager.getAllInfoByStudentID ()

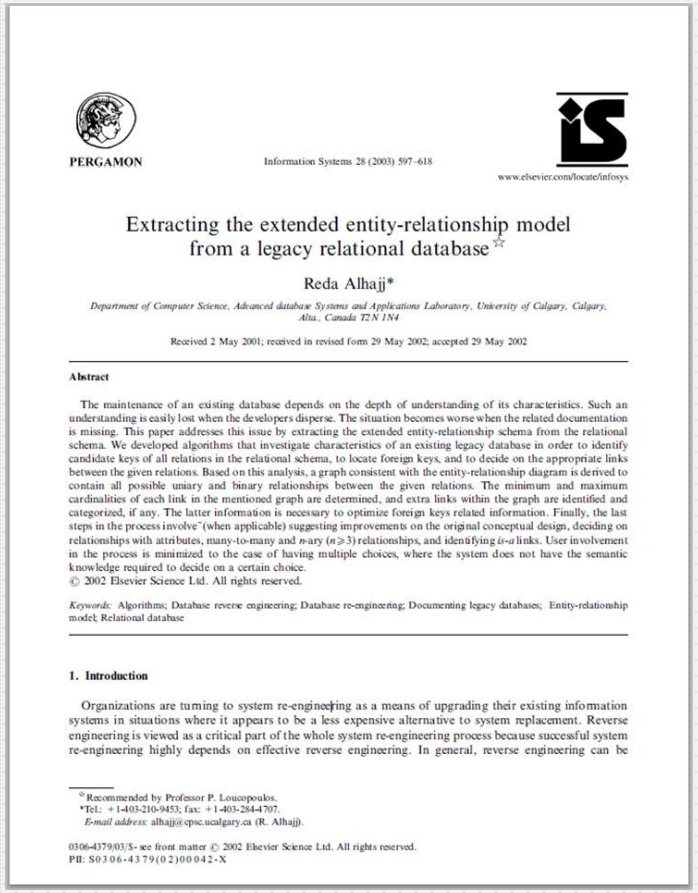
getSTLogin()

getSTDetails()

Challenges

- How the model is described by a schema
- How the database is used in the source code

Related works



PERGAMON

Information Systems 28 (2003) 597–618

www.elsevier.com/locate/infosys

Extracting the extended entity-relationship model from a legacy relational database ☆

Reda Alhajj*

Department of Computer Science, Advanced database Systems and Applications Laboratory, University of Calgary, Calgary, Alta., Canada T2N 1N4

Received 2 May 2001; received in revised form 29 May 2002; accepted 29 May 2002

Abstract

The maintenance of an existing database depends on the depth of understanding of its characteristics. Such an understanding is easily lost when the developers disperse. The situation becomes worse when the related documentation is missing. This paper addresses this issue by extracting the extended entity-relationship schema from the relational schema. We developed algorithms that investigate characteristics of an existing legacy database in order to identify candidate keys of all relations in the relational schema, to locate foreign keys, and to decide on the appropriate links between the given relations. Based on this analysis, a graph consistent with the entity-relationship diagram is derived to contain all possible uniary and binary relationships between the given relations. The minimum and maximum cardinalities of each link in the mentioned graph are determined, and extra links within the graph are identified and categorized, if any. The latter information is necessary to optimize foreign keys related information. Finally, the last steps in the process involve (when applicable) suggesting improvements on the original conceptual design, deciding on relationships with attributes, many-to-many and n-ary ($n \geqslant 3$) relationships, and identifying *is-a* links. User involvement in the process is minimized to the case of having multiple choices, where the system does not have the semantic knowledge required to decide on a certain choice.

Keywords: Algorithms; Database reverse engineering; Database re-engineering; Documenting legacy databases; Entity-relationship model; Relational database

1. Introduction

Organizations are turning to system re-engineering as a means of upgrading their existing information systems in situations where it appears to be a less expensive alternative to system replacement. Reverse engineering is viewed as a critical part of the whole system re-engineering process because successful system re-engineering highly depends on effective reverse engineering. In general, reverse engineering can be

☆ Recommended by Professor P. Loucopoulos.

*Tel.: +1-403-210-9453; fax: +1-403-284-4707.
E-mail address: alhajj@cpsc.ucalgary.ca (R. Alhajj).

0306-4379/03/$- see front matter © 2002 Elsevier Science Ltd. All rights reserved.
PII: S0306-4379(02)00042-X

Related works

PERGAMON

Information Systems 28 (2003) 597–618

www.elsevier.com/locate/infosys

Extracting the extended entity-relationship model from a legacy relational database ☆

Reda Alhajj*

Department of Computer Science, Advanced database Systems and Applications Laboratory, University of Calgary, Calgary, Alta., Canada T2N 1N4

Received 2 May 2001; received in revised form 29 May 2002; accepted 29 May 2002

Abstract

The maintenance of an existing database depends on the depth of understanding of its characteristics. Such an understanding is easily lost when the developers disperse. The situation becomes worse when the related documentation is missing. This paper addresses this issue by extracting the extended entity-relationship schema from the relational schema. We developed algorithms that investigate characteristics of an existing legacy database in order to identify candidate keys of all relations in the relational schema, to locate foreign keys, and to decide on the appropriate links between the given relations. Based on this analysis, a graph consistent with the entity-relationship diagram is derived to contain all possible uniary and binary relationships between the given relations. The minimum and maximum cardinalities of each link in the mentioned graph are determined, and extra links within the graph are identified and categorized, if any. The latter information is necessary to optimize foreign keys related information. Finally, the last steps in the process involve (when applicable) suggesting improvements on the original conceptual design, deciding on relationships with attributes, many-to-many and n-ary ($n \geqslant 3$) relationships, and identifying *is-a* links. User involvement in the process is minimized to the case of having multiple choices, where the system does not have the semantic knowledge required to decide on a certain choice.

Keywords: Algorithms; Database reverse engineering; Database re-engineering; Documenting legacy databases; Entity-relationship model; Relational database

1. Introduction

Organizations are turning to system re-engineering as a means of upgrading their existing information systems in situations where it appears to be a less expensive alternative to system replacement. Reverse engineering is viewed as a critical part of the whole system re-engineering process because successful system re-engineering highly depends on effective reverse engineering. In general, reverse engineering can be

☆ Recommended by Professor P. Loucopoulos.

*Tel.: +1-403-210-9453; fax: +1-403-284-4707.
E-mail address: alhajj@cpsc.ucalgary.ca (R. Alhajj).

0306-4379/03/$- see front matter © 2002 Elsevier Science Ltd. All rights reserved.
PII: S0306-4379(02)00042-X

Proceedings 12th IEEE Int. Conf. on Data Engineering ICDE'96, New-Orleans (USA), February 26 - March 1, 1996, IEEE Press, pp. 218-227.

Towards the Reverse Engineering of Denormalized Relational Databases

J-M. Petit, F. Toumani, J-F. Boulicaut, J. Kouloumdjian
Laboratoire d'Ingénierie des Systèmes d'Information
INSA Lyon, 20 av. Albert Einstein, Bât. 501
F-69621 Villeurbanne cedex
e-mail: jean-marc.petit@lisi.insa-lyon.fr

Abstract

This paper describes a method to cope with denormalized relational schemas in a database reverse engineering process. We propose two main steps to improve the understanding of data semantics. Firstly we extract inclusion dependencies by analyzing the equi-join queries embedded in application programs and by querying the database extension. Secondly we show how to discover only functional dependencies which influence the way attributes should be restructured. The method is interactive since an expert user has to validate the presumptions on the elicited dependencies. Moreover, a restructuring phase leads to a relational schema in third normal form provided with key constraints and referential integrity constraints. Finally, we sketch how an Entity-Relationship schema can be derived from such information.

1 Introduction

The aim of a Database Reverse Engineering (DBRE) process is to improve the understanding of the data semantics. Many aspects of database evolution, especially for old databases where data semantics has been lost for years, require a DBRE process [7]. Such current situations are the re-engineering of the so-called legacy systems or the federation of distributed databases. Many works have already been done where a conceptual schema (often based on an extension of the Entity-Relationship (ER) model [4]) is derived from a hierarchical database [15, 2], a network database [2] or a relational database [3, 15, 13, 2, 21, 5]. A DBRE process is naturally split into two major steps [18]:

- Eliciting the data semantics from the existing system

  Various sources of information can be relevant for tackling this task, e.g., the physical schema, the database extension, the application programs, but especially expert users.

- Expressing the extracted semantics with a high level data model

  This task consists in a schema translation activity and gives rise to several difficulties since the concepts of the original model do not overlap those of the target model.

In the context of relational databases, most of the DBRE methods [15, 13, 21] focus only on the schema translation task since they assume that the constraints (e.g., functional dependencies or foreign keys) are available at the beginning of the process. However, to cope with real-life situations, such strong assumptions are not realistic since old versions of DataBase Management Systems (DBMSs) do not support such declarations.

Some recent works [19, 22, 1, 16] have proposed independently to alleviate the assumptions on the knowledge available a priori. Given a schema in third Normal Form (3NF), the key idea is to fetch the needed information from the data manipulation statements embedded in application programs. We have already interesting results in this direction [16, 17, 18]. Unlike [5], we do not constrain the relational schema with a *consistent naming of key attributes* and unlike [13, 21, 10, 9], we do not need to have all the structural constraints before applying the method.

A current assumption in existing DBRE methods, including our previous results, is to impose the relational schema to be in 3NF to ensure that each relation corresponds to a unique object of the application domain. Nevertheless, Johannesson has shown that several objects, the so-called *hidden objects*, can be encoded in a 3NF relation [10]. He introduces a formal framework to handle such cases in a DBRE process. Unlike Johannesson who still has strong assumptions

1063-6382/96 $5.00 © 1996 IEEE

Related works

PERGAMON

Information Systems 28 (2003) 597–618

www.elsevier.com/locate/infosys

Extracting the extended entity-relationship model from a legacy relational database ☆

Reda Alhajj*

Department of Computer Science, Advanced database Systems and Applications Laboratory, University of Calgary, Calgary, Alta., Canada T2N 1N4

Received 2 May 2001; received in revised form 29 May 2002; accepted 29 May 2002

Abstract

The maintenance of an existing database depends on the depth of understanding of its characteristics. Such an understanding is easily lost when the developers disperse. The situation becomes worse when the related documentation is missing. This paper addresses this issue by extracting the extended entity-relationship schema from the relational schema. We developed algorithms that investigate characteristics of an existing legacy database in order to identify candidate keys of all relations in the relational schema, to locate foreign keys, and to decide on the appropriate links between the given relations. Based on this analysis, a graph consistent with the entity-relationship diagram is derived to contain all possible uniary and binary relationships between the given relations. The minimum and maximum cardinalities of each link in the mentioned graph are determined, and extra links within the graph are identified and categorized, if any. The latter information is necessary to optimize foreign keys related information. Finally, the last steps in the process involve (when applicable) suggesting improvements on the original conceptual design, deciding on relationships with attributes, many-to-many and n-ary ($n \geqslant 3$) relationships, and identifying *is-a* links. User involvement in the process is minimized to the case of having multiple choices, where the system does not have the semantic knowledge required to decide on a certain choice.

Keywords: Algorithms; Database reverse engineering; Database re-engineering; Documenting legacy databases; Entity-relationship model; Relational database

1. Introduction

Organizations are turning to system re-engineering as a means of upgrading their existing information systems in situations where it appears to be a less expensive alternative to system replacement. Reverse engineering is viewed as a critical part of the whole system re-engineering process because successful system re-engineering highly depends on effective reverse engineering. In general, reverse engineering can be

☆ Recommended by Professor P. Loucopoulos.

*Tel.: +1-403-210-9453; fax: +1-403-284-4707.
E-mail address: alhajj@cpsc.ucalgary.ca (R. Alhajj).

0306-4379/03/$ - see front matter © 2002 Elsevier Science Ltd. All rights reserved.
PII: S0306-4379(02)00042-X

Proceedings 12th IEEE Int. Conf. on Data Engineering ICDE'96, New-Orleans (USA), February 26 - March 1, 1996, IEEE Press, pp. 218-227.

Towards the Reverse Engineering of Denormalized Relational Databases

J-M. Petit, F. Toumani, J-F. Boulicaut, J. Kouloumdjian
Laboratoire d'Ingénierie des Systèmes d'Information
INSA Lyon, 20 av. Albert Einstein, Bât. 501
F-69621 Villeurbanne cedex
e-mail: jean-marc.petit@lisi.insa-lyon.fr

Abstract

This paper describes a method to cope with denormalized relational schemas in a database reverse engineering process. We propose two main steps to improve the understanding of data semantics. Firstly we extract inclusion dependencies by analyzing the equi-join queries embedded in application programs and by querying the database extension. Secondly we show how to discover only functional dependencies which influence the way attributes should be restructured. The method is interactive since an expert user has to validate the presumptions on the elicited dependencies. Moreover, a restructuring phase leads to a relational schema in third normal form provided with key constraints and referential integrity constraints. Finally, we sketch how an Entity-Relationship schema can be derived from such information.

1 Introduction

The aim of a Database Reverse Engineering (DBRE) process is to improve the understanding of the data semantics. Many aspects of database evolution, especially for old databases where data semantics has been lost for years, require a DBRE process [7]. Such current situations are the re-engineering of the so-called legacy systems or the federation of distributed databases. Many works have already been done where a conceptual schema (often based on an extension of the Entity-Relationship (ER) model [4]) is derived from a hierarchical database [15, 2], a network database [2] or a relational database [3, 15, 13, 2, 21, 5]. A DBRE process is naturally split into two major steps [18]:

- Eliciting the data semantics from the existing system

  Various sources of information can be relevant for tackling this task, e.g., the physical schema, the database extension, the application programs, but especially expert users.

- Expressing the extracted semantics with a high level data model

  This task consists in a schema translation activity and gives rise to several difficulties since the concepts of the original model do not overlap those of the target model.

In the context of relational databases, most of the DBRE methods [15, 13, 21] focus only on the schema translation task since they assume that the constraints (e.g., functional dependencies or foreign keys) are available at the beginning of the process. However, to cope with real-life situations, such strong assumptions are not realistic since old versions of DataBase Management Systems (DBMSs) do not support such declarations.

Some recent works [19, 22, 1, 16] have proposed independently to alleviate the assumptions on the knowledge available a priori. Given a schema in third Normal Form (3NF), the key idea is to fetch the needed information from the data manipulation statements embedded in application programs. We have already interesting results in this direction [16, 17, 18]. Unlike [5], we do not constrain the relational schema with a *consistent naming of key attributes* and unlike [13, 21, 10, 9], we do not need to have all the structural constraints before applying the method.

A current assumption in existing DBRE methods, including our previous results, is to impose the relational schema to be in 3NF to ensure that each relation corresponds to a unique object of the application domain. Nevertheless, Johannesson has shown that several objects, the so-called *hidden objects*, can be encoded in a 3NF relation [10]. He introduces a formal framework to handle such cases in a DBRE process. Unlike Johannesson who still has strong assumptions

1063-6382/96 $5.00 © 1996 IEEE

An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications

Dong Qiu Bixin Li
Southeast University, China
{dongqiu, bx.li}@seu.edu.cn

Zhendong Su
University of California, Davis, USA
su@cs.ucdavis.edu

ABSTRACT

Modern database applications are among the most widely used and complex software systems. They constantly evolve, responding to changes to data, database schemas, and code. It is challenging to manage these changes and ensure that everything co-evolves consistently. For example, when a database schema is modified, all the code that interacts with the database must be changed accordingly. Although database evolution and software evolution have been extensively studied in isolation, the co-evolution of schema and code has largely been unexplored.

This paper presents the first comprehensive empirical analysis of the co-evolution of database schemas and code in ten popular large open-source database applications, totaling over 160K revisions. Our major findings include: 1) Database schemas evolve frequently during the application lifecycle, exhibiting a variety of change types with similar distributions across the studied applications; 2) Overall, schema changes induce significant code-level modifications, while certain change types have more impact on code than others; and 3) Co-change analyses can be viable to automate or assist with database application evolution. We have also observed that: 1) 80% of the schema changes happened in 20-30% of the tables, while nearly 40% of the tables did not change; and 2) Referential integrity constraints and stored procedures are rarely used in our studied subjects. We believe that our study reveals new insights into how database applications evolve and useful guidelines for designing assistive tools to aid their evolution.

Categories and Subject Descriptors

H.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; H.2.1 [**Database Management**]: Logical Design—*Schema and subschema*

General Terms

Language, Measurement

Keywords

Co-evolution, database application, empirical analysis

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia

1. INTRODUCTION

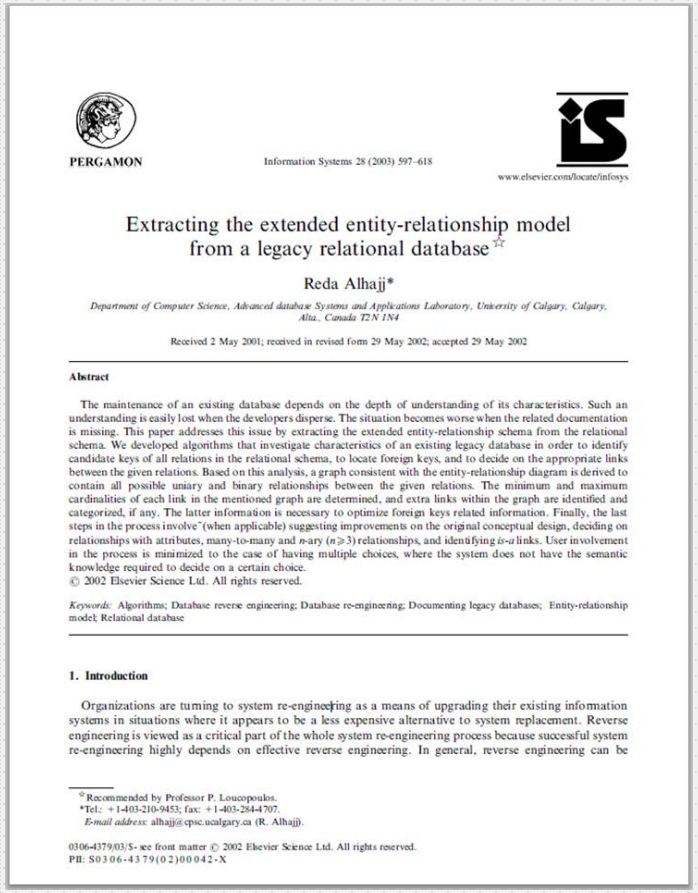
A database application is a software system that collects, manages, and retrieves data, which are typically stored in a database managed by a database management system (DBMS) and organized *w.r.t.* database schemas. For example, most online services are powered by database applications. Wikis, social networking systems (SNS), Web-based content management systems (CMS), mailing systems, enterprise resource planning systems (ERP) are all database applications. As Figure 1 illustrates, a program needs to obey the structure of the data organization defined by a schema when it accesses the data. Namely, a schema is a mediator that manages the interactions between code and data, bridging their gap.

Software systems are subject to continuous evolution due to modified system requirements; database applications are no exception. Cleve *et al.* [5] observe that little work exists on understanding the evolution of database applications considering both data and code. Different from traditional applications, the evolution of database applications is more complex. For example, consider a system that uses a table USER to store both user authentication information and other personal data. Now the system requirements change, and the system needs to store user authentication information and personal data separately. Thus, the original table USER must be split into two new tables, say USER_LOGIN and USER_DETAILS. Data and application code must be synchronized to be consistent with the new schemas. First, the original data organization should be migrated to the new one defined by USER_LOGIN and USER_DETAILS. Second, the original application code that accesses data in USER must be modified to correctly access the newly organized data in USER_LOGIN and USER_DETAILS.

Figure 1 illustrates these two types of co-evolution in database applications: 1) data co-evolve with schemas, and 2) code co-evolves with schemas. The first type of co-evolution involves three main tasks: i) predicting and estimating the effects before the proposed schema changes are performed; ii) rewriting the existing DBMS-level queries to work on the new schemas; and iii) migrating data to the new schemas. The second type involves two main tasks: i) evaluating the cost of reconciling the existing code *w.r.t.* the new schemas before any schema changes; and ii) locating and modifying all impacted code regions after applying the schema changes.

The database community has addressed the first co-evolution problem gracefully to support automatic data migration and DBMS-level query rewriting to operate on the new schemas [6, 7]. However, little work has considered the second co-evolution problem. Its difficulties are twofold. First, query updates and data migration for the first problem are done by DB Administrators (DBA), who have the domain knowledge. In contrast, the application developers, who have different level of database knowledge, may not precisely capture the whole evolution process of the database structure. In

Related works



PERGAMON

Information Systems 28 (2003) 597–618

www.elsevier.com/locate/infosys

Extracting the extended entity-relationship model from a legacy relational database ☆

Reda Alhajj*

Department of Computer Science, Advanced database Systems and Applications Laboratory, University of Calgary, Calgary, Alta., Canada T2N 1N4

Received 2 May 2001; received in revised form 29 May 2002; accepted 29 May 2002

Abstract

The maintenance of an existing database depends on the depth of understanding of its characteristics. Such an understanding is easily lost when the developers disperse. The situation becomes worse when the related documentation is missing. This paper addresses this issue by extracting the extended entity-relationship schema from the relational schema. We developed algorithms that investigate characteristics of an existing legacy database in order to identify candidate keys of all relations in the relational schema, to locate foreign keys, and to decide on the appropriate links between the given relations. Based on this analysis, a graph consistent with the entity-relationship diagram is derived to contain all possible uniary and binary relationships between the given relations. The minimum and maximum cardinalities of each link in the mentioned graph are determined, and extra links within the graph are identified and categorized, if any. The latter information is necessary to optimize foreign keys related information. Finally, the last steps in the process involve (when applicable) suggesting improvements on the original conceptual design, deciding on relationships with attributes, many-to-many and n-ary ($n \geqslant 3$) relationships, and identifying *is-a* links. User involvement in the process is minimized to the case of having multiple choices, where the system does not have the semantic knowledge required to decide on a certain choice.

Keywords: Algorithms; Database reverse engineering; Database re-engineering; Documenting legacy databases; Entity-relationship model; Relational database

1. Introduction

Organizations are turning to system re-engineering as a means of upgrading their existing information systems in situations where it appears to be a less expensive alternative to system replacement. Reverse engineering is viewed as a critical part of the whole system re-engineering process because successful system re-engineering highly depends on effective reverse engineering. In general, reverse engineering can be

☆ Recommended by Professor P. Loucopoulos.

*Tel.: +1-403-210-9453; fax: +1-403-284-4707.
E-mail address: alhajj@cpsc.ucalgary.ca (R. Alhajj).

0306-4379/03/$- see front matter © 2002 Elsevier Science Ltd. All rights reserved.
PII: S0306-4379(02)00042-X

Proceedings 12th IEEE Int. Conf. on Data Engineering ICDE'96, New-Orleans (USA), February 26 - March 1, 1996, IEEE Press, pp. 218-227.

Towards the Reverse Engineering of Denormalized Relational Databases

J-M. Petit, F. Toumani, J-F. Boulicaut, J. Kouloumdjian
Laboratoire d'Ingénierie des Systèmes d'Information
INSA Lyon, 20 av. Albert Einstein, Bât. 501
F-69621 Villeurbanne cedex
e-mail: jean-marc.petit@lisi.insa-lyon.fr

Abstract

This paper describes a method to cope with denormalized relational schemas in a database reverse engineering process. We propose two main steps to improve the understanding of data semantics. Firstly we extract inclusion dependencies by analyzing the equi-join queries embedded in application programs and by querying the database extension. Secondly we show how to discover only functional dependencies which influence the way attributes should be restructured. The method is interactive since an expert user has to validate the presumptions on the elicited dependencies. Moreover, a restructuring phase leads to a relational schema in third normal form provided with key constraints and referential integrity constraints. Finally, we sketch how an Entity-Relationship schema can be derived from such information.

1 Introduction

The aim of a Database Reverse Engineering (DBRE) process is to improve the understanding of the data semantics. Many aspects of database evolution, especially for old databases where data semantics has been lost for years, require a DBRE process [7]. Such current situations are the re-engineering of the so-called legacy systems or the federation of distributed databases. Many works have already been done where a conceptual schema (often based on an extension of the Entity-Relationship (ER) model [4]) is derived from a hierarchical database [15, 2], a network database [2] or a relational database [3, 15, 13, 2, 21, 5]. A DBRE process is naturally split into two major steps [18]:

- Eliciting the data semantics from the existing system

  Various sources of information can be relevant for tackling this task, e.g., the physical schema, the database extension, the application programs, but especially expert users.

- Expressing the extracted semantics with a high level data model

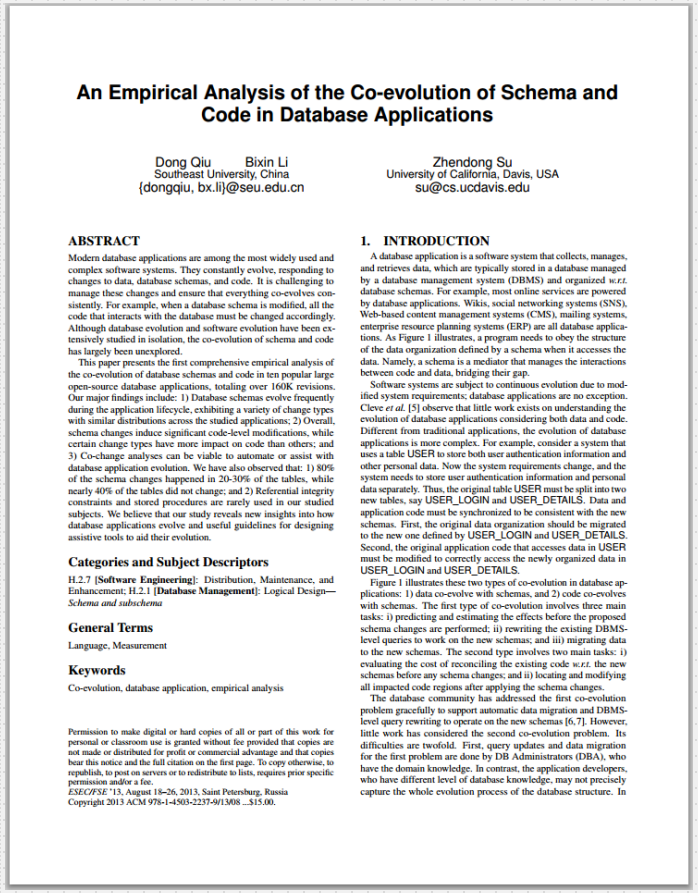
  This task consists in a schema translation activity and gives rise to several difficulties since the concepts of the original model do not overlap those of the target model.

In the context of relational databases, most of the DBRE methods [15, 13, 21] focus only on the schema translation task since they assume that the constraints (e.g., functional dependencies or foreign keys) are available at the beginning of the process. However, to cope with real-life situations, such strong assumptions are not realistic since old versions of DataBase Management Systems (DBMSs) do not support such declarations.

Some recent works [19, 22, 1, 16] have proposed independently to alleviate the assumptions on the knowledge available a priori. Given a schema in third Normal Form (3NF), the key idea is to fetch the needed information from the data manipulation statements embedded in application programs. We have already interesting results in this direction [16, 17, 18]. Unlike [5], we do not constrain the relational schema with a *consistent naming of key attributes* and unlike [13, 21, 10, 9], we do not need to have all the structural constraints before applying the method.

A current assumption in existing DBRE methods, including our previous results, is to impose the relational schema to be in 3NF to ensure that each relation corresponds to a unique object of the application domain. Nevertheless, Johannesson has shown that several objects, the so-called *hidden objects*, can be encoded in a 3NF relation [10]. He introduces a formal framework to handle such cases in a DBRE process. Unlike Johannesson who still has strong assumptions

1063-6382/96 $5.00 © 1996 IEEE



An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications

Dong Qiu Bixin Li
Southeast University, China
{dongqiu, bx.li}@seu.edu.cn

Zhendong Su
University of California, Davis, USA
su@cs.ucdavis.edu

ABSTRACT

Modern database applications are among the most widely used and complex software systems. They constantly evolve, responding to changes to data, database schemas, and code. It is challenging to manage these changes and ensure that everything co-evolves consistently. For example, when a database schema is modified, all the code that interacts with the database must be changed accordingly. Although database evolution and software evolution have been extensively studied in isolation, the co-evolution of schema and code has largely been unexplored.

This paper presents the first comprehensive empirical analysis of the co-evolution of database schemas and code in ten popular large open-source database applications, totaling over 160K revisions. Our major findings include: 1) Database schemas evolve frequently during the application lifecycle, exhibiting a variety of change types with similar distributions across the studied applications; 2) Overall, schema changes induce significant code-level modifications, while certain change types have more impact on code than others; and 3) Co-change analyses can be viable to automate or assist with database application evolution. We have also observed that: 1) 80% of the schema changes happened in 20-30% of the tables, while nearly 40% of the tables did not change; and 2) Referential integrity constraints and stored procedures are rarely used in our studied subjects. We believe that our study reveals new insights into how database applications evolve and useful guidelines for designing assistive tools to aid their evolution.

Categories and Subject Descriptors

H.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; H.2.1 [**Database Management**]: Logical Design—*Schema and subschema*

General Terms

Language, Measurement

Keywords

Co-evolution, database application, empirical analysis

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia

1. INTRODUCTION

A database application is a software system that collects, manages, and retrieves data, which are typically stored in a database managed by a database management system (DBMS) and organized *w.r.t.* database schemas. For example, most online services are powered by database applications. Wikis, social networking systems (SNS), Web-based content management systems (CMS), mailing systems, enterprise resource planning systems (ERP) are all database applications. As Figure 1 illustrates, a program needs to obey the structure of the data organization defined by a schema when it accesses the data. Namely, a schema is a mediator that manages the interactions between code and data, bridging their gap.

Software systems are subject to continuous evolution due to modified system requirements; database applications are no exception. Cleve *et al.* [5] observe that little work exists on understanding the evolution of database applications considering both data and code. Different from traditional applications, the evolution of database applications is more complex. For example, consider a system that uses a table USER to store both user authentication information and other personal data. Now the system requirements change, and the system needs to store user authentication information and personal data separately. Thus, the original table USER must be split into two new tables, say USER_LOGIN and USER_DETAILS. Data and application code must be synchronized to be consistent with the new schemas. First, the original data organization should be migrated to the new one defined by USER_LOGIN and USER_DETAILS. Second, the original application code that accesses data in USER must be modified to correctly access the newly organized data in USER_LOGIN and USER_DETAILS.

Figure 1 illustrates these two types of co-evolution in database applications: 1) data co-evolve with schemas, and 2) code co-evolves with schemas. The first type of co-evolution involves three main tasks: i) predicting and estimating the effects before the proposed schema changes are performed; ii) rewriting the existing DBMS-level queries to work on the new schemas; and iii) migrating data to the new schemas. The second type involves two main tasks: i) evaluating the cost of reconciling the existing code *w.r.t.* the new schemas before any schema changes; and ii) locating and modifying all impacted code regions after applying the schema changes.

The database community has addressed the first co-evolution problem gracefully to support automatic data migration and DBMS-level query rewriting to operate on the new schemas [6, 7]. However, little work has considered the second co-evolution problem. Its difficulties are twofold. First, query updates and data migration for the first problem are done by DB Administrators (DBA), who have the domain knowledge. In contrast, the application developers, who have different level of database knowledge, may not precisely capture the whole evolution process of the database structure. In

No previous work has been done to understand database documentation practices at source code level.

Goal

How Do Developers Document Database Usages in Source Code?

Methodology

GitHub

381,161 projects

Methodology

GitHub

381,161 projects

Identified the projects using SQL

18,828 projects

Methodology

GitHub

381,161 projects

Identified the projects using SQL

18,828 projects

 ≥ 1  ≥ 1

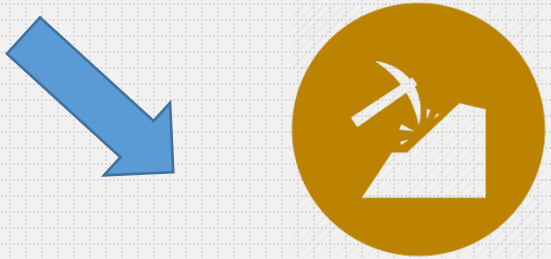
3,113 projects

Methodology

A survey

GitHub

3,113 projects



A mining-based analysis

Methodology

GitHub

3,113 projects

A survey

with

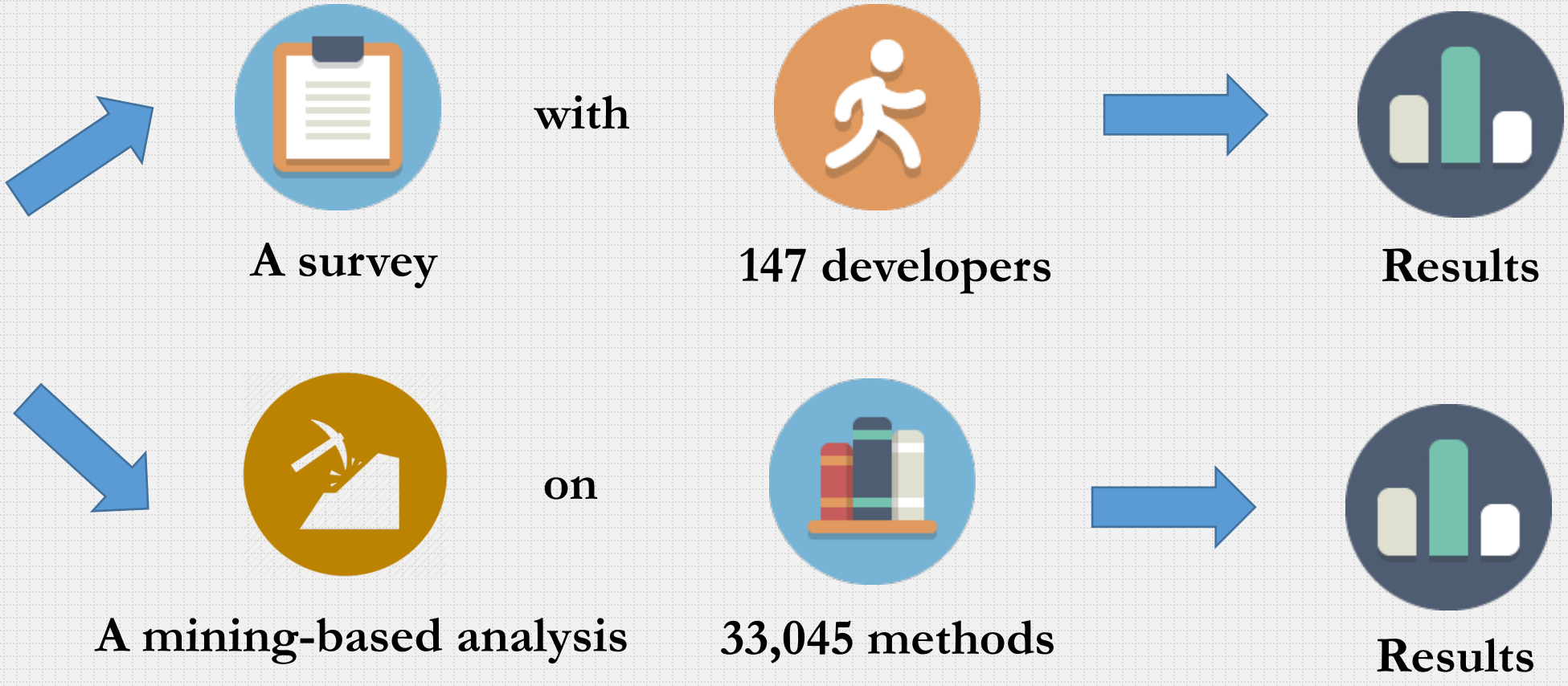
147 developers

A mining-based analysis

on

33,045 methods

Methodology



Research Questions

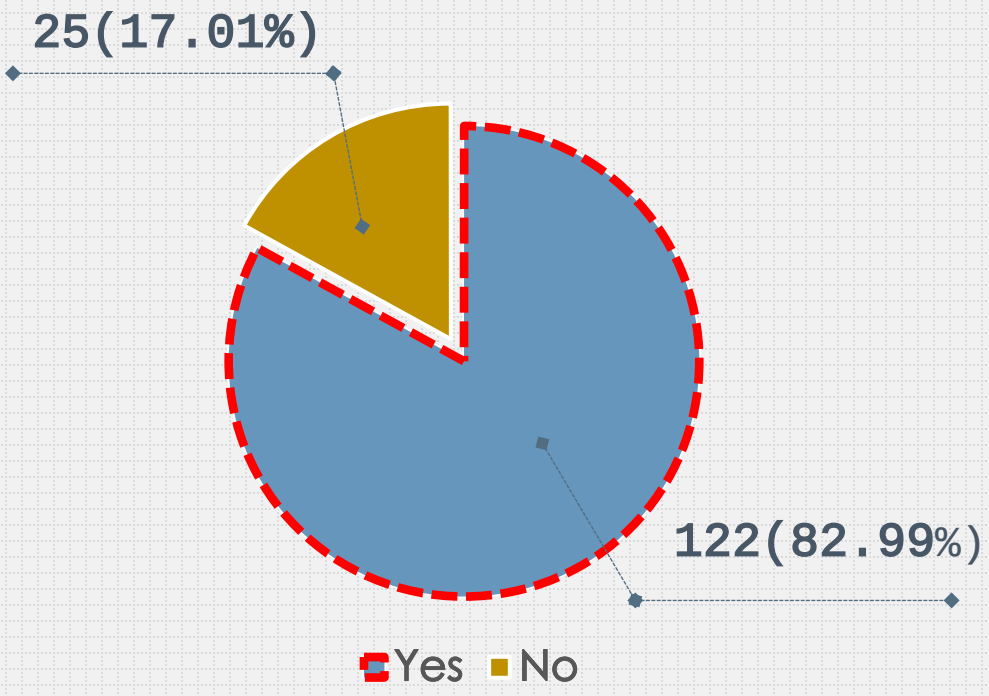
RQ1. Do developers document database-related methods

RQ2. Do developers update comments for database-related methods

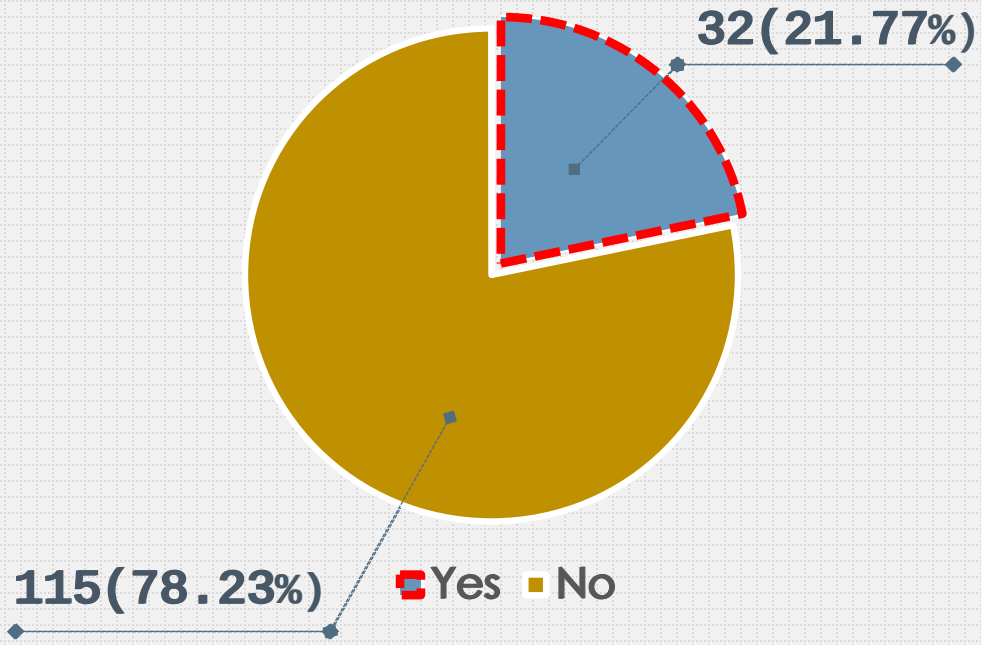
RQ3. How difficult is to understand the database schema constraints along call-chains

RQ1. Do developers comment methods in source code that locally execute SQL queries and statements?

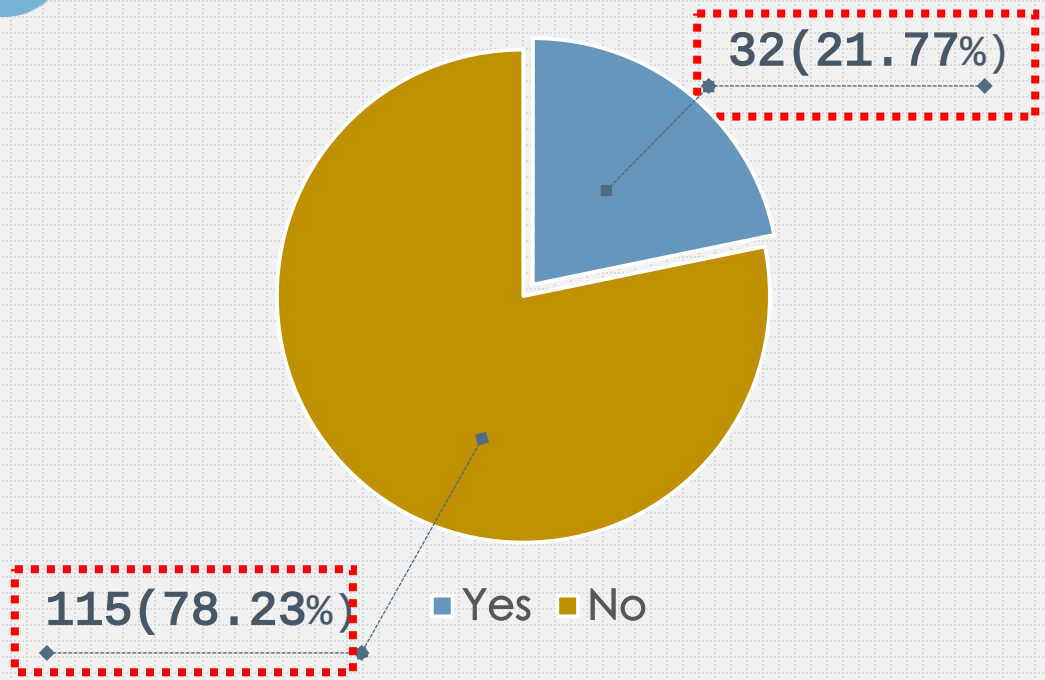
SQ1. Do you add/write documentation comments to methods in the source code?



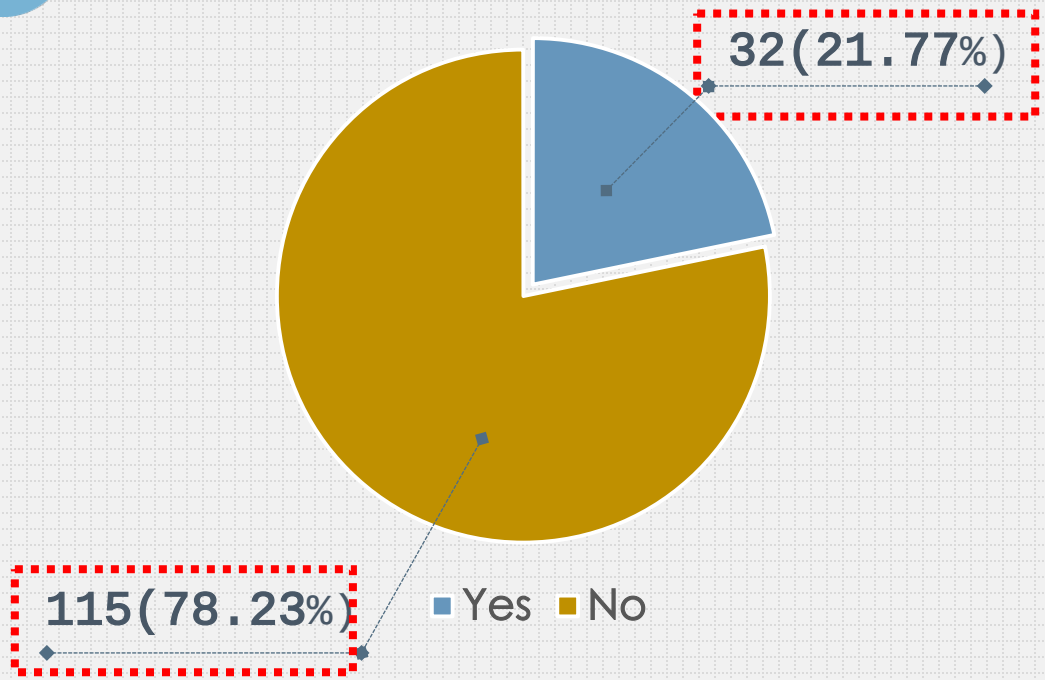
SQ2. Do you write source code comments detailing database schema constraints?



RQ1. Do developers comment methods in source code that locally execute SQL queries and statements?



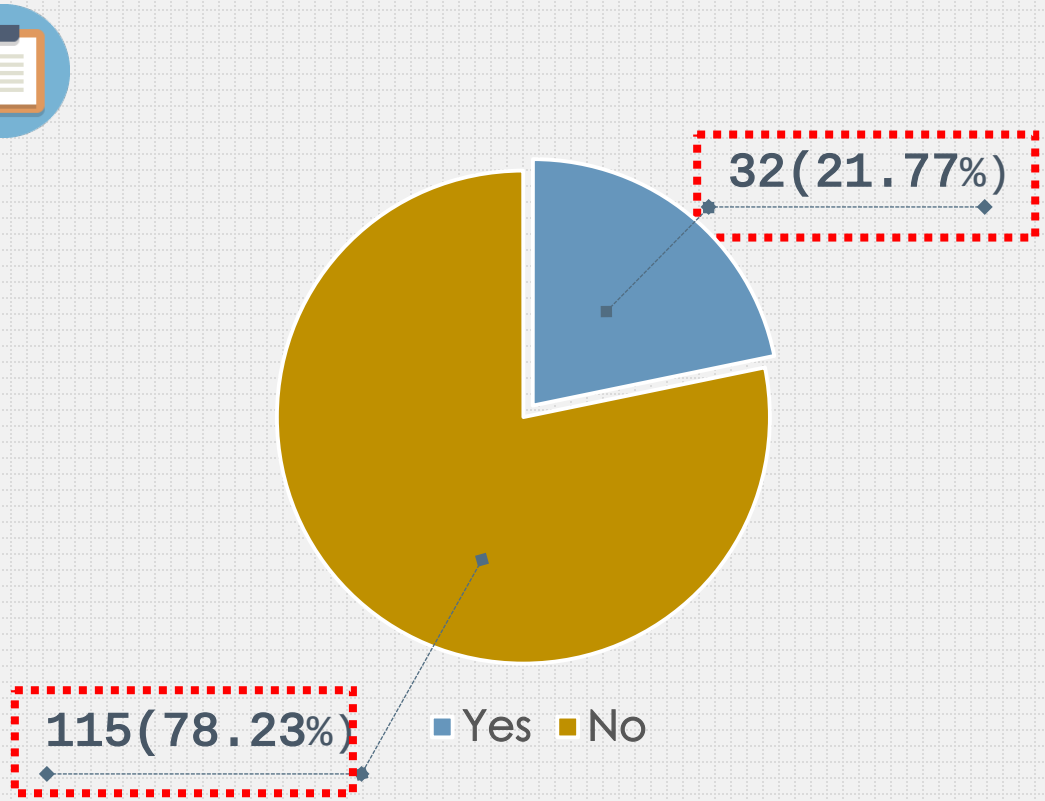
RQ1. Do developers comment methods in source code that locally execute SQL queries and statements?

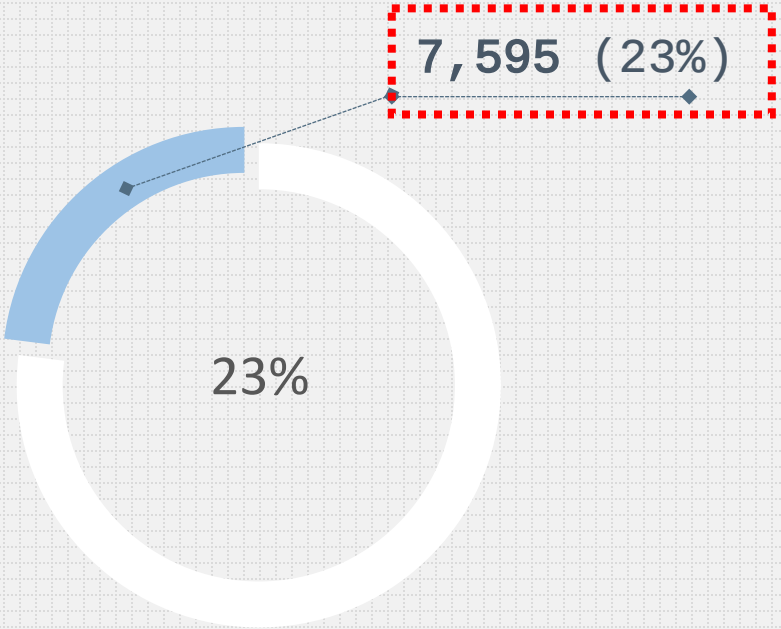


> *"The database schema and documentation takes care of that. I can always look at the table definition very easily."*

> *"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."*

RQ1. Do developers comment methods in source code that locally execute SQL queries and statements?

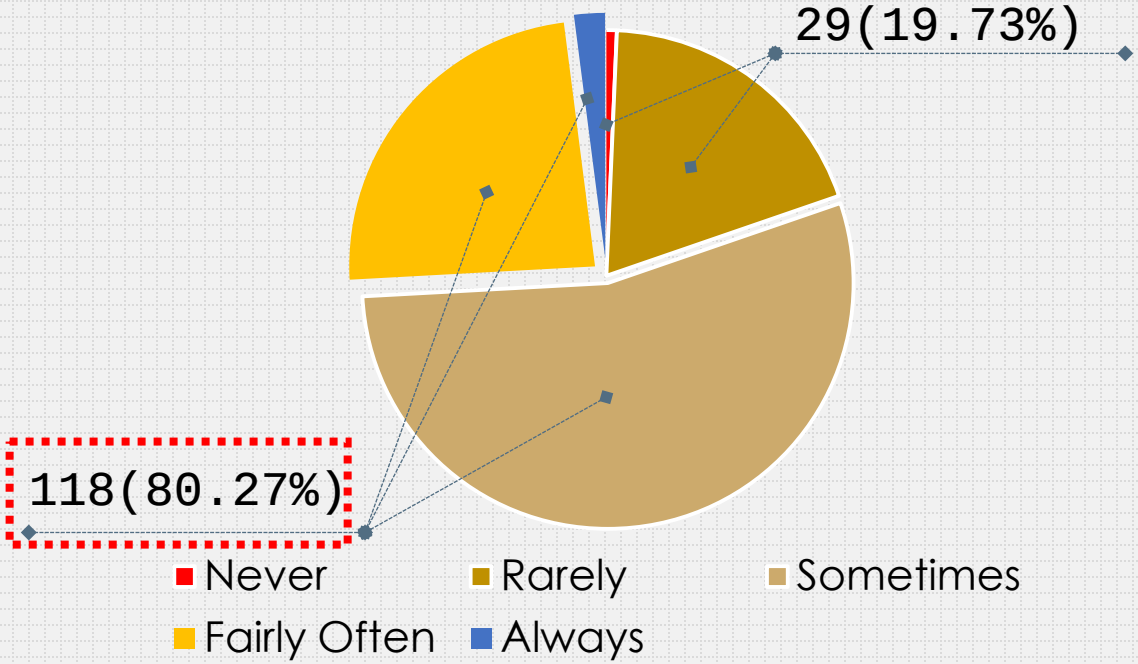




In the 3,113 projects, we identified a total of 33,045 methods invoking SQL ueries/statements.

RQ2. Do developers update comments of database-related methods during the evolution of a system?

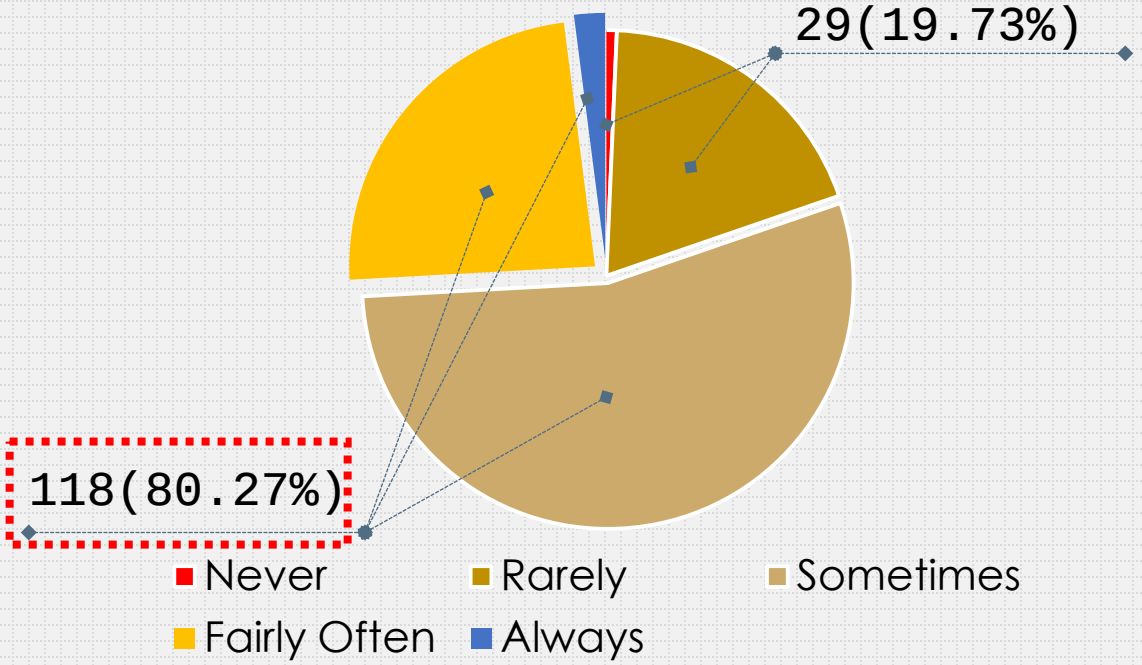
SQ3. How often do you find outdated comments in source code?



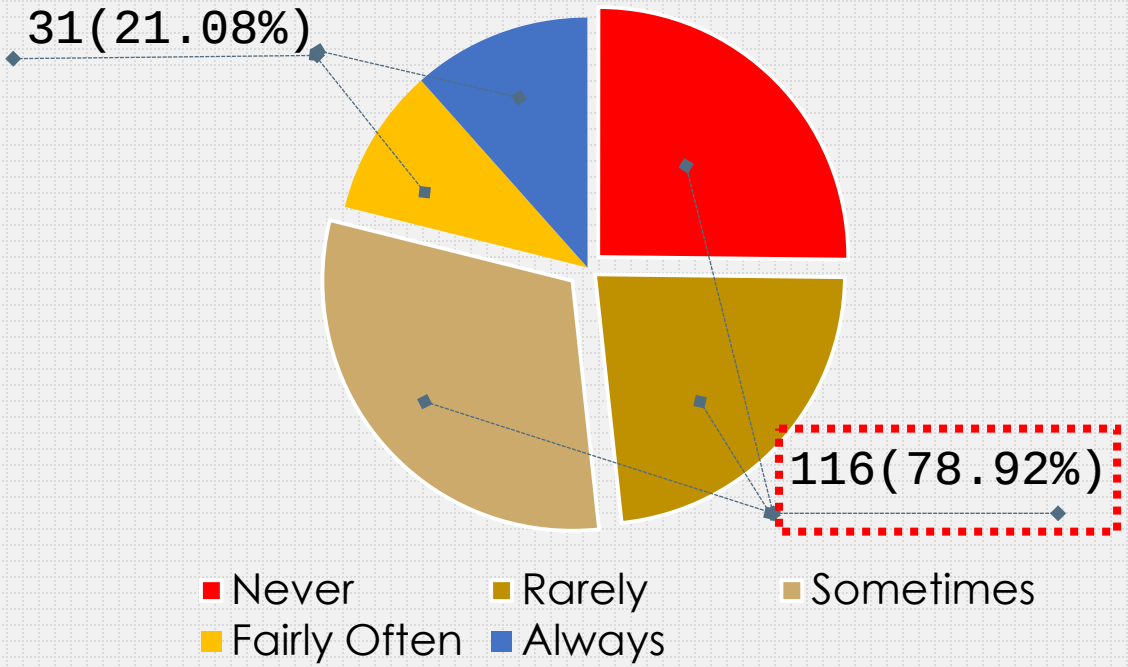
SQ4. When you make changes to database related methods, how often do you update comments?

RQ2. Do developers update comments of database-related methods during the evolution of a system?

SQ3. How often do you find outdated comments in source code?



SQ4. When you make changes to database related methods, how often do you update comments?



RQ2. Do developers update comments of database-related methods during the evolution of a system?

3,113 projects

8.5% Had explicit releases

264 projects

RQ2. Do developers update comments of database-related methods during the evolution of a system?

3,113 projects

8.5% Had explicit releases

264 projects

2,662 methods (Invoke SQL)

RQ2. Do developers update comments of database-related methods during the evolution of a system?

3,113 projects

8.5% Had explicit releases

264 projects

2,662 methods (Invoke SQL)

23.2% were updated

618 methods

RQ2. Do developers update comments of database-related methods during the evolution of a system?

3,113 projects

8.5% Had explicit releases

264 projects

2,662 methods (Invoke SQL)

23.2% were updated

618 methods

82.8% didn't updated comments

512 methods

RQ2. Do developers update comments of database-related methods during the evolution of a system?

3,113 projects

8.5% Had explicit releases

264 projects

2,662 methods (Invoke SQL)

23.2% were updated

618 methods

82.8% didn't updated comments

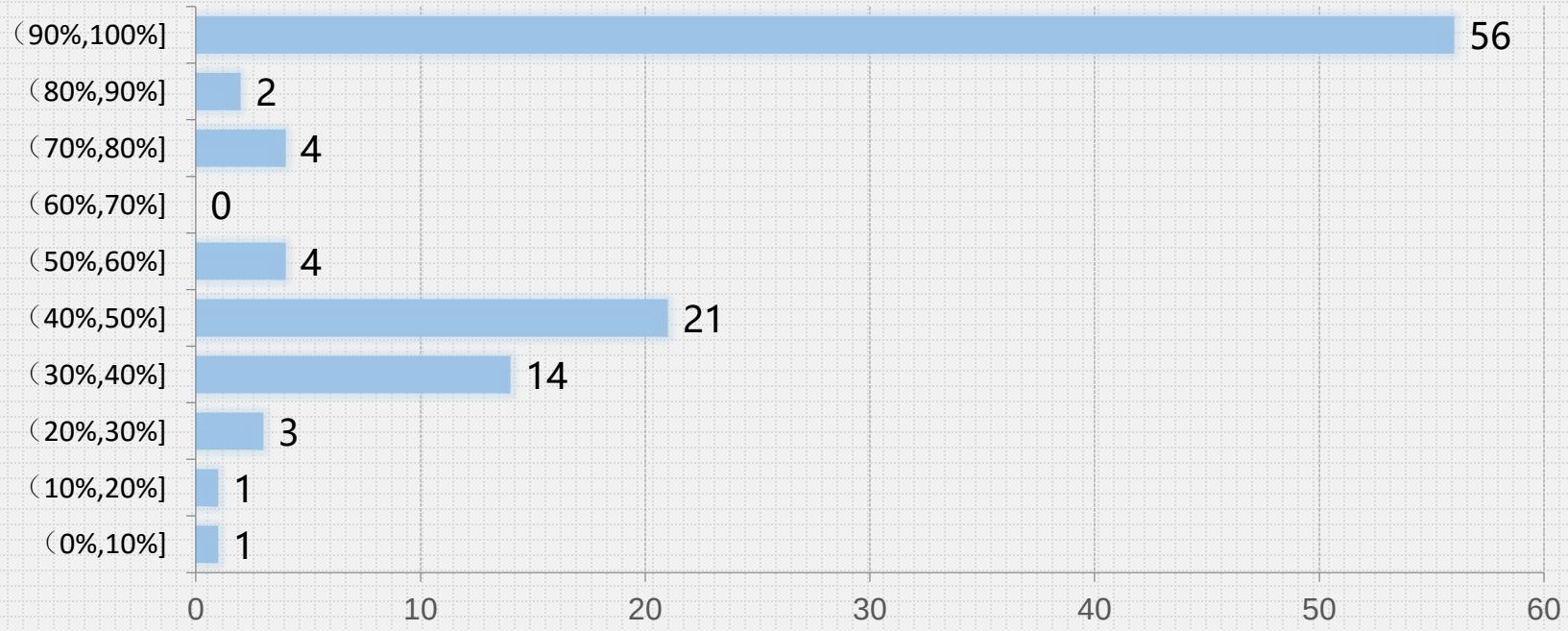
17.2% updated comments

512 methods

106 methods

RQ2. Do developers update comments of database-related methods during the evolution of a system?

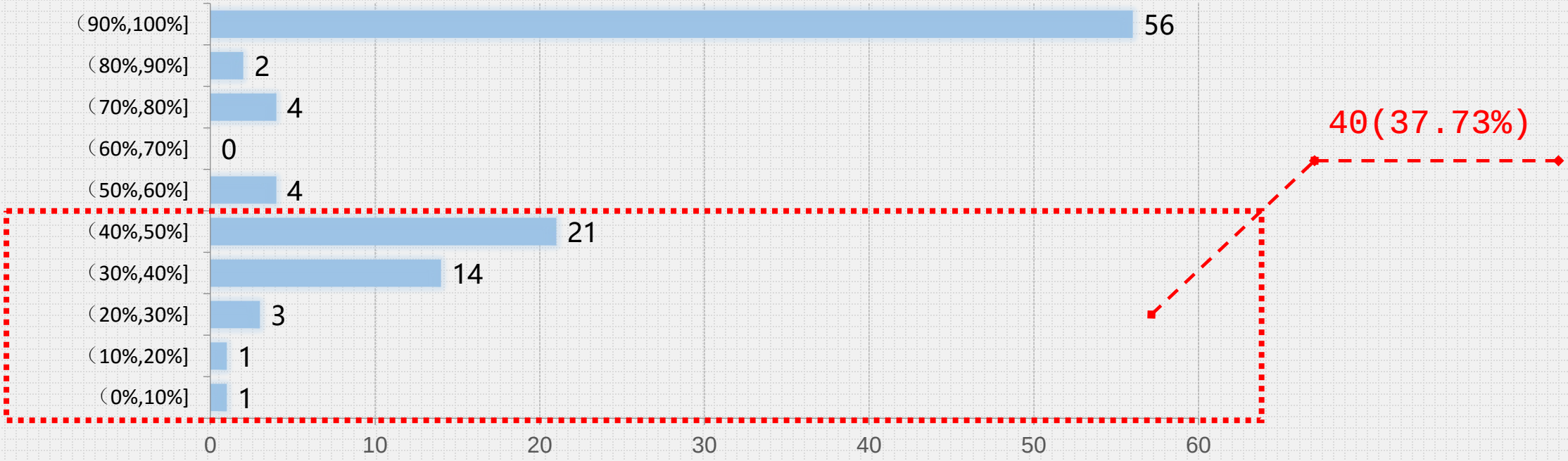
106 methods



frequency that the comments were updated when the method was modified

RQ2. Do developers update comments of database-related methods during the evolution of a system?

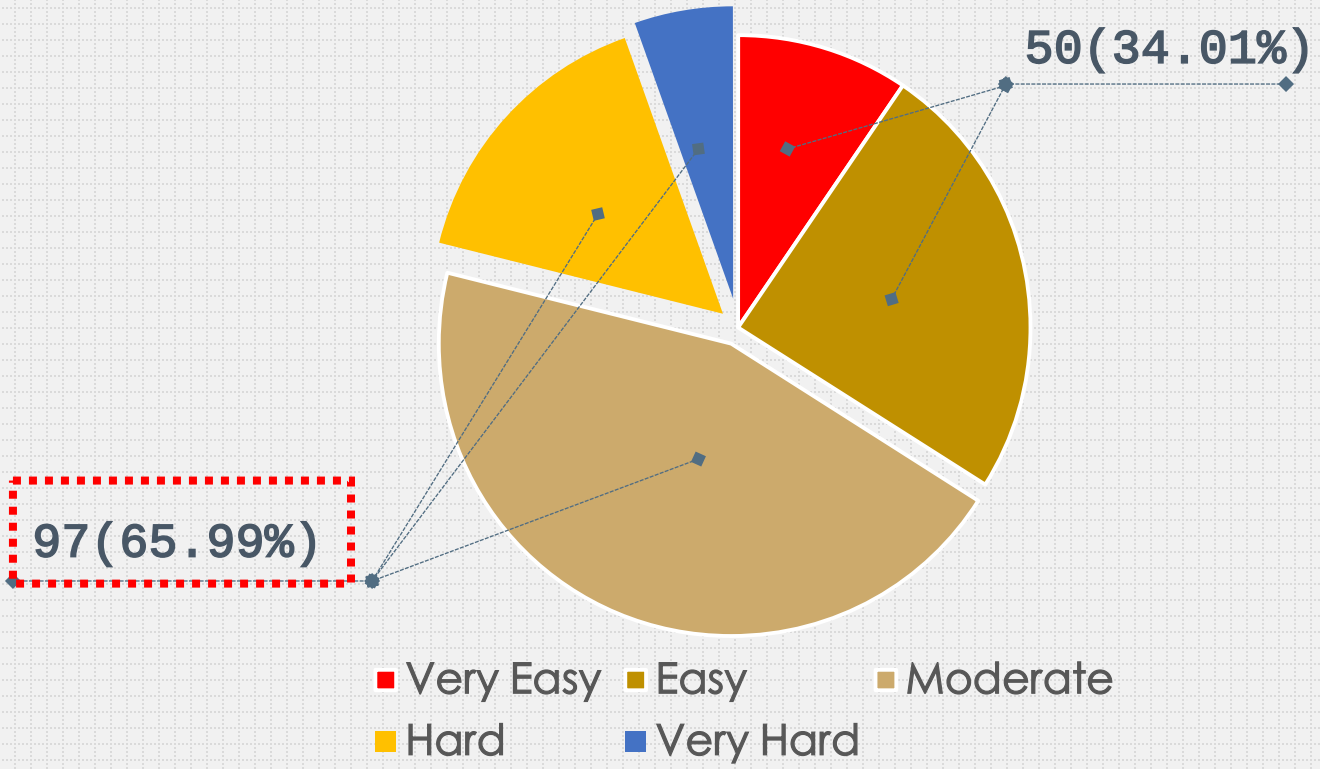
106 methods



frequency that the comments were updated when the method was modified

RQ3. How difficult is it for developers to understand propagated schema constraints along call-chains?

SQ5. 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



Lessons learnt

(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

(iii) Tracing schema constraints through call-chains in the call graph is not an easy task in most of the cases

Lessons learnt

(i) Documenting database usages and constraints is not a common practice in source code methods

Documentation

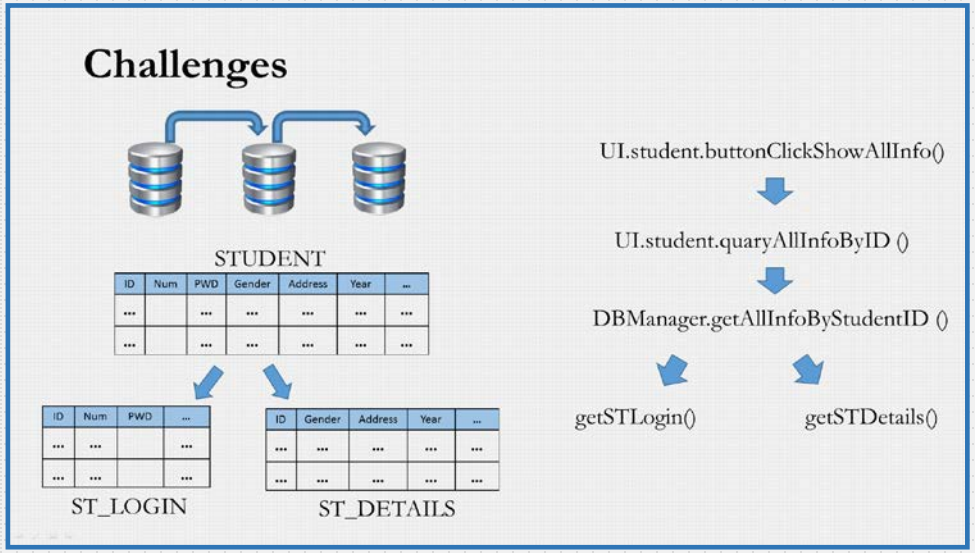
(ii) Developers do not update comments when changes are done to database-related methods

Automation

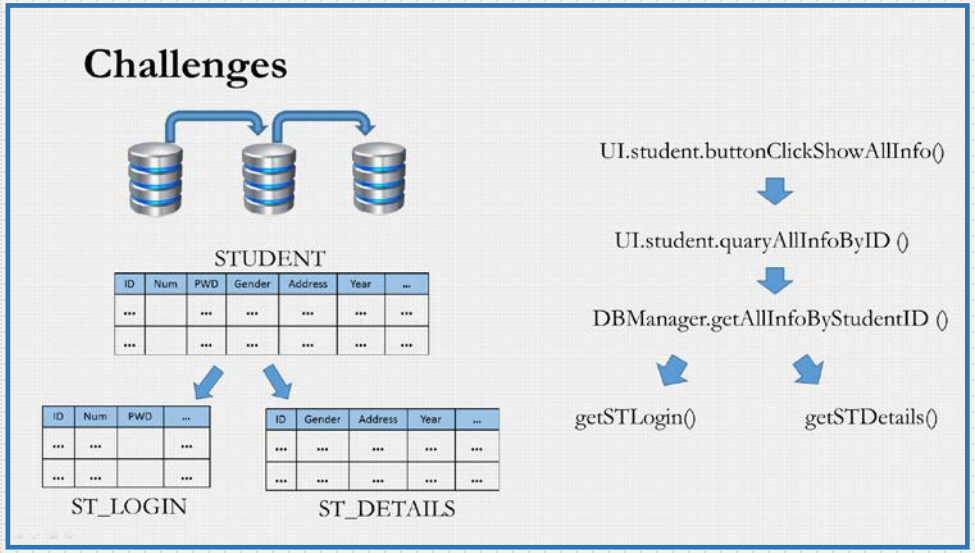
(iii) Tracing schema constraints through call-chains in the call graph is not an easy task in most of the cases

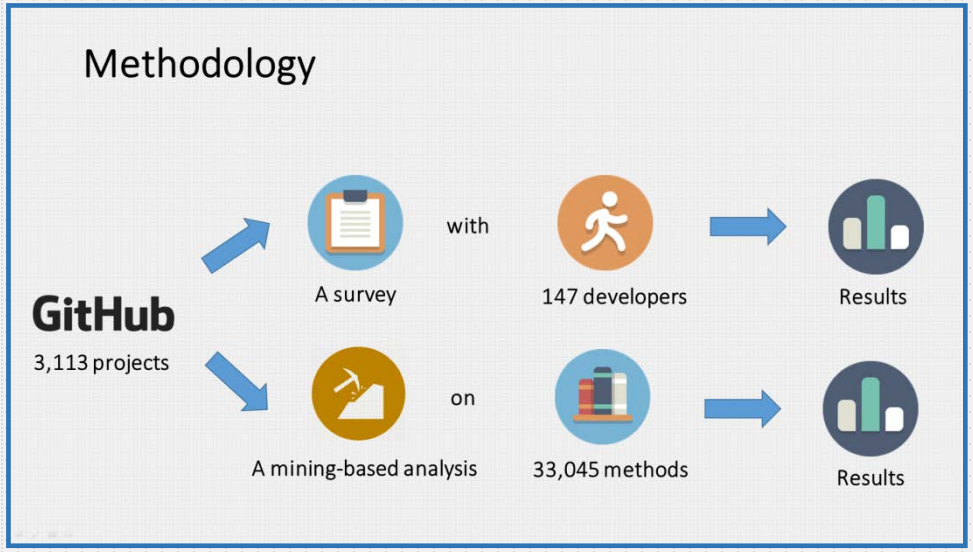
Calling context

Summary


Challenges
STUDENT
ID Num PWD Gender Address Year ...
ST_LOGIN
ID Num PWD ...
ST_DETAILS
ID Gender Address Year ...
UI.student.buttonClickShowAllInfo()
UI.student.quaryAllInfoByID ()
DBManager.getAllInfoByStudentID ()
getSTLogin()
getSTDetails()

Summary


Challenges
STUDENT
ID Num PWD Gender Address Year ...
ST_LOGIN
ST_DETAILS
UI.student.buttonClickShowAllInfo()
UI.student.quaryAllInfoByID ()
DBManager.getAllInfoByStudentID ()
getSTLogin()
getSTDetails()


Methodology
GitHub
3,113 projects
A survey
with
147 developers
Results
A mining-based analysis
on
33,045 methods
Results

Summary

Challenges

STUDENT

ID	Num	PWD	Gender	Address	Year	...
...		...	...	...	...	...
...		...	...	...	...	...

ID	Num	PWD	...
...	...		...
...	...		...

ST_LOGIN

ID	Gender	Address	Year	...
...	...	...	...	...
...	...	...	...	...

ST_DETAILS

UI.student.buttonClickShowAllInfo()

↓

UI.student.quaryAllInfoByID ()

↓

DBManager.getAllInfoByStudentID ()

↓

getSTLogin() getSTDetails()

Methodology

Research Questions

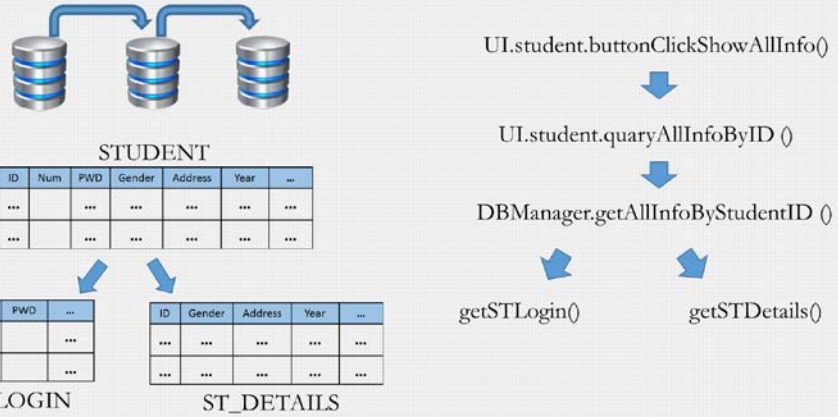
RQ1. Do developers document database-related methods

RQ2. Do developers update database-related methods

RQ3. How difficult is to understand database schema along call-chains

Summary

Challenges



Methodology

Research Questions

RQ1. Do developers document database-related methods

RQ2. Do developers update database-related methods

RQ3. How difficult is to understand database schema along call-chains

Lessons learnt

(i) Documenting database usages and constraints is not a common practice in source code methods	**Documentation**
(ii) Developers do not update comments when changes are done to database-related methods	**Automation**
(iii) Tracing schema constraints through call-chains in the call graph is not an easy task in most of the cases	**Calling context**