Towards a Survival Analysis of Database Framework Usage in Java Projects

Mathieu Goeminne and Tom Mens
Software Engineering Lab, University of Mons, Belgium

***Abstract*—Many software projects rely on a relational database in order to realize part of their functionality. Various database frameworks and object-relational mappings have been developed and used to facilitate data manipulation. Little is known about whether and how such frameworks co-occur, how they complement or compete with each other, and how this changes over time. We empirically studied these aspects for 5 Java database frameworks, based on a corpus of 3,707 *GitHub* Java projects. In particular, we analysed whether certain database frameworks co-occur frequently, and whether some database frameworks get replaced over time by others. Using the statistical technique of survival analysis, we explored the survival of the database frameworks in the considered projects. This provides useful evidence to software developers about which frameworks can be used successfully in combination and which combinations should be avoided.**

I. Introduction

Many software projects are relying on databases for the proper functioning of the application. To facilitate this data management and manipulation, a wide variety of database frameworks has been proposed and used, especially for software projects developed in popular languages such as Java. These frameworks typically introduce a language-specific abstraction layer to avoid hardcoding and manually adapting SQL queries to any changes occurring in the database schema.

Software developers occasionally replace database frameworks used in their projects, or introduce extra database frameworks that offer additional functionality. In order to help developers cope with this phenomenon, it is necessary to study which database frameworks are used together and how they interact over time.

In this paper, we shed more light on this framework usage in open source Java projects, by carrying out an empirical study on the evolution of a corpus of Java projects in *GitHub* that use *relational* database technology. Despite the rising popularity of more recent technologies such as NoSQL, we focus on *relational databases*, because they are still omnipresent in current-day software projects and because more historical data is available to analyze their usage.

Our longitudinal study of database framework usage in Java projects addresses the following research questions:
RQ_0: Which database frameworks are most popular?
RQ_1: Which combinations of database frameworks "co-occur" in the projects in which they are used?
RQ_2: How long do database frameworks "survive" in the projects in which they occur?
RQ_3: Does the introduction of a database framework influence the survivability of another one?

II. Survival Analysis

Many of these research questions are clearly related to the time-dependent nature of the projects being analyzed, and the occurrence of specific "events" during the projects lifetime (such as the introduction or disappearance of a particular database framework). To answer these questions in a statistically valid way, we therefore resort to the statistical technique of survival analysis [1], [2]. We rely on the CRAN packages survival for computation and ggplot2 for visualization.

Survival analysis models the time it takes for a specific event (such as the disappearance of a particular database framework in a Java project at some point in time) to occur. The technique allows to take into account *right-censored* data, for which it may be unknown whether the event occurred or not because the subject has "disappeared". For example, we have no precise idea of when a particular database framework will disappear from a Java project if the database framework is still present in the project at the last day of the considered period of study.

Since we cannot assume a particular distribution of survival times, we need to resort to non-parametric methods such as the Kaplan-Meier estimator [3]. A survival function models the probability of an arbitrary subject in the dataset to survive t units of time after the start of the study. A Kaplan-Meier curve visualizes the cumulative probability to survive, or, more generally, that an event occurs. It starts at value 1 (100% probability of survival at time zero) and decreases monotonically over time. In this study, the observed event is the definitive disappearance of a framework from a project.

To test whether there is a difference with statistical significance between two survival distributions we use the survdiff function that implements the Mantel-Haenszel test [4].

III. Related Work

In [5], we empirically analyzed the evolution of the usage of SQL, Hibernate and JPA in a single large open source Java project. The current paper carries out a macro-level study on thousands of projects.

Chen et al. [6] proposed a static code analysis framework for detecting and fixing performance issues and potential bugs in ORM usage. Their analysis revealed that the modifications made after analysis caused an important improvement of the studied systems' response time. Maule et al. [7] studied a commercial object-oriented content management system to statically analyze the impact of database schema changes on the source code. Qiu et al. [8] empirical analyzed the co-evolution of database schemas and code in ten open-source

978-1-4673-7532-0/15 © 2015 IEEE

Accepted for publication by IEEE. © 2015 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

database applications from various domains. Whereas our focus is on Java projects, they studied specific change types inside the database schema and the impact of such changes on PHP code.

The statistical technique of *survival analysis* used in this paper has been employed by other software engineering researchers as well. Samoladas et al. [9] predicted the survivability of open source projects over time. Scanniello [10] analyzed dead code in five open source Java software systems. Kyriakakis et al. [11] studied function usage and function removal in five large PHP applications. Claes et al. [12] studied the survival of installability conflicts in Debian packages.

IV. Data Extraction

We focus on *open source* projects only since we need full access to the source code development history of the studied projects. We analyse *Java* projects because Java is one of the most popular programming languages today. More specifically, we study projects taken from the *Github Java Corpus* proposed by Allamanis and Sutton [13]. They processed *GitHub* events stored by *Github Archive* (www.githubarchive.org) and only retained projects marked as Java projects. In order to get a *quality* corpus, they removed all projects that were never forked according to *GitHub*. They also compared the *id*s of commits in order to manually remove projects that are very likely (undeclared) forks of another project. This filtering decreases the probability to obtain strongly related individuals in the considered project population, and hence reduces the risk to obtain statistical results biased by overrepresented (groups of) projects. Among the 14,765 projects proposed in the *Github Java Corpus*, 13,307 (90.1%) projects have still an available Git repository on 24 March 2015. We considered these Java projects as potential candidates for our empirical analysis, and created a local clone of each of them.

We considered 19 Java database frameworks as potential candidates for our study. These frameworks need to have a direct means for accessing the database. The frameworks were selected by skimming recent scientific publications, Stack Exchange and blog posts. As an additional constraint, since our goal is to study the evolution over time of database framework usage in Java projects, we only consider frameworks of at least 3 years old. Although our list is not exhaustive, it covers the most frequently cited frameworks.

As a baseline, we also included JDBC in our study. Unlike other considered frameworks, it doesn't provide any abstraction of a database schema and forces the developers to write SQL queries. JDBC is still heavily used because such a low level connection allows to submit complex queries that would be difficult or impossible to express with a higher level database framework.

We determined the presence of each framework in each Java project by analyzing the import statements in Java files, as well as the presence of specific configuration files (e.g., for Hibernate). For each commit of each considered Java project, we retraced a historical view of the files that can be related to a particular framework.

V. Empirical Analysis

This section addresses our research questions by means of tables, visualizations and statistical tests.

RQ_0: Which database frameworks are most popular?

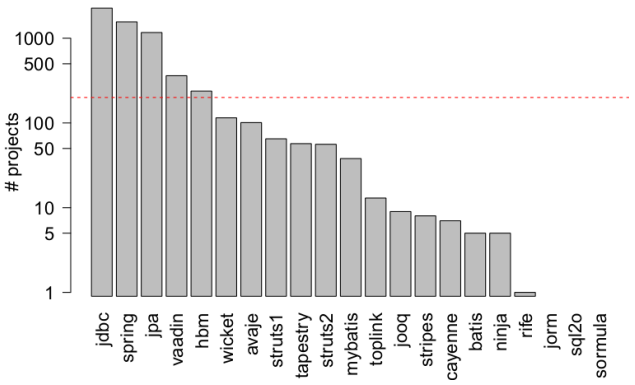

Fig. 1. Number of projects (in log scale) in which a given database framework occurs. Threshold 200 shown in red.

Fig. 1 shows the number of considered Java projects in which the considered database frameworks occur. We observe a high imbalance. Only 5 frameworks (including JDBC), summarized in Table I, occur in more than 200 distinct projects. Of all considered active projects from *Github Java Corpus*, only 3,707 Java projects used at least one of these 5 frameworks. Only these 5 frameworks will be analyzed for the remaining research questions, because for the other frameworks we do not have sufficient occurrences to obtain statistically significant results.

TABLE I
Selected Java database frameworks.

Framework name	URL	Occurs if the project contains at least a file	#projects
JDBC	www.oracle.com/technetwork/java/javase/jdbc	importing java.sql	2,271
Spring	projects.spring.io/spring-framework	importing org.springframework	1,562
JPA	www.tutorialspoint.com/jpa	importing javax.persistence	1,168
Vaadin-GWT	vaadin.com	importing com.google.gwt	361
Hibernate	hibernate.org	whose name ends with .hbm.xml	238

The *Spring* framework aims to facilitate the implementation of a standard structure in Java applications. An optional *Spring* extension based on JDBC and an object-relational mapping can provide access to relational and NoSQL databases.

JPA is a Java API for describing the relation between a Java entity and its mapped database element. Several frameworks, including *Hibernate*, can exploit this description for providing such a service. Because the frameworks that actually use these annotations cannot always be determined, we don't consider

JPA annotations as an indicator of the use of any framework but JPA itself.

Vaadin is a framework for developing web applications. It introduces the notion of *domain layer*, which abstracts the database structure through Java classes hosting the business logic of the application.

RQ_1: Which combinations of database frameworks "co-occur" in the projects in which they are used?

We identified which of the 5 considered database frameworks occurred throughout the lifetime of each considered project, and we computed all possible intersections of framework occurrence in Fig. 2.

JDBC occurs as the only database framework in 56.3% of all projects. At the other side of the spectrum, Hibernate occurs as the only framework in only 2.9% of all projects. If we look at their intersection, the large majority (82.8%) of all projects that have used Hibernate have also used JDBC during their lifetime.

Something similar can be observed for JDBC and JPA. JPA occurs in isolation in 29.5% of all projects, while almost half of all projects that have used JPA (49.3% to be precise) have also used JDBC during their lifetime.

Similarly, when comparing Hibernate and JPA, we observe that 49.6% of all projects that have used Hibernate have also used JPA, while 44.1% of all projects that have used Hibernate have also used JPA and JDBC.

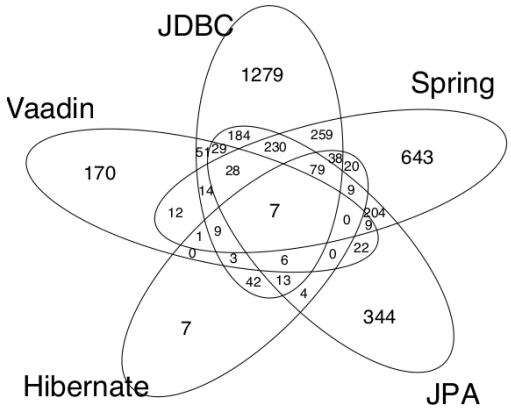

Fig. 2. Number of Java projects using a given number of database frameworks (over the entire project's lifetime).

These high numbers could be due to the fact that some database frameworks are used as supporting technologies for others (e.g., Spring typically uses JDBC for database access), while some frameworks are complementing each other (e.g., Vaadin has an optional module called JPAContainer for supporting JPA annotations).

To determine for which frameworks this is the case, we studied the *"co-occurrence"* of different frameworks within the same project. This happens when files relating to both frameworks are present in at least one of the project's commits (but typically in many more commits).

Table II shows vertically the number of projects having used a given number of distinct frameworks over their entire history, and horizontally the maximum number of distinct "co-occurring" frameworks. Almost all values reside on the diagonal, implying that in the large majority of all cases (97.5%, i.e., 1213/1273), **different database frameworks used in a project tend to co-occur**.

TABLE II
NUMBER OF PROJECTS INVOLVING A GIVEN NUMBER OF FRAMEWORKS, OVER THEIR ENTIRE LIFETIME AND IN CO-OCCURRENCE.

# co-occurring fw. → ↓ total # frameworks used	1	2	3	4	5
1	2,443				
2	22	776			
3	2	16	328		
4	0	0	18	104	
5	0	0	1	1	5

Focusing on specific combinations of frameworks, Table III reports the number of projects in which two database frameworks co-occurred at least once during the project's lifetime. Not surprisingly, we observe that JDBC frequently co-occurs with other frameworks. That lets us suppose that JDBC is used as a supporting technology that provides services not offered by the other frameworks. 80.1% of all projects that used Hibernate have also used JDBC in co-occurrence; 48.4% of all projects that used JPA have used JDBC in co-occurrence; 41.3% of all projects that used Spring have used JDBC in co-occurrence; and 39.6% of all projects that used Vaadin have used JDBC in co-occurrence.

TABLE III
NUMBER OF PROJECTS IN WHICH PAIRS OF DATABASE FRAMEWORKS CO-OCCUR.

	Spring	JPA	Vaadin	Hibernate
JDBC	645	565	143	192
Spring		558	76	156
JPA			98	105
Vaadin				22

Some database frameworks seem to complement one another. For example, 47.8% of all projects using Spring also use JPA. Other database frameworks appear to be in competition. For example, Vaadin co-occurs with Hibernate in only 22 projects, which makes up 9.2% of all projects using Hibernate, and only 6.1% of all projects using Vaadin. To a lesser extent, Vaadin also co-occurs infrequently together with JPA or Spring.

RQ_2: How long do database frameworks "survive" in the projects in which they occur?

Fig. 3 shows the Kaplan-Meier survival curves of the selected frameworks. After their introduction, **all database frameworks remain present in more than 45% of the projects**. Nevertheless, **we observe different trends in framework survivability**. For example, in 11.7% of all cases Hibernate is removed 30 days after its introduction. In the same time interval, Spring is only removed from 3.7% of the projects. Three years after its

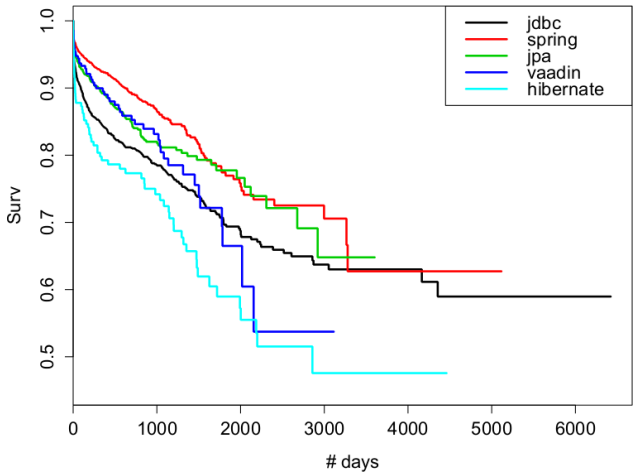

Fig. 3. Survival curves of database framework occurrence in the considered projects.

introduction, Hibernate disappears from 27.6% of all projects, while Spring disappears from 14.5% of all projects in the same interval.

TABLE IV
P-VALUES OF TESTS FOR DIFFERENCE OF SURVIVAL RATES BETWEEN TWO DATABASE FRAMEWORKS.

$A \rightarrow$ $\downarrow B$	Spring	JPA	Vaadin	Hibernate
JDBC	$<$ **0.001 [-]**	**0.001 [-]**	0.242	0.010
Spring	—	0.030	0.017	$<$ **0.001 [+]**
JPA		—	0.427	$<$ **0.001 [+]**
Vaadin			—	0.017

Table IV shows the p-values of the Mantel-Haenszel tests to check for a difference in survival rates for each pair of frameworks. Significant results are shown in boldface, based on significance level $\alpha = \frac{0.05}{10}$ after Bonferroni correction since we perform 10 comparisons [14]. Based on visual comparison of the survival curves, we marked with [+] if framework B has a significantly better survival rate than framework A, and [-] in the opposite case. We observe that **JPA and Spring have higher survival rates than JDBC and Hibernate.**

RQ_3: Does the introduction of a database framework influence the survivability of another one?

In order to determine if the introduction of a framework B influences the survival of a framework A already present in the same project, we computed two survival curves C_1 and C_2. C_1 is based on all projects in which A and B co-occurred, while B was introduced in the project after A. C_2 considers all projects that have used A while B never co-occurred together with A. We performed a visual comparison of C_1 (shown in red dashed lines) and C_2 (shown in black straight lines) on all pairs of database frameworks.

Fig. 4 compares the survival of Spring and JDBC. Introducing Spring when JDBC is already present seems to improve

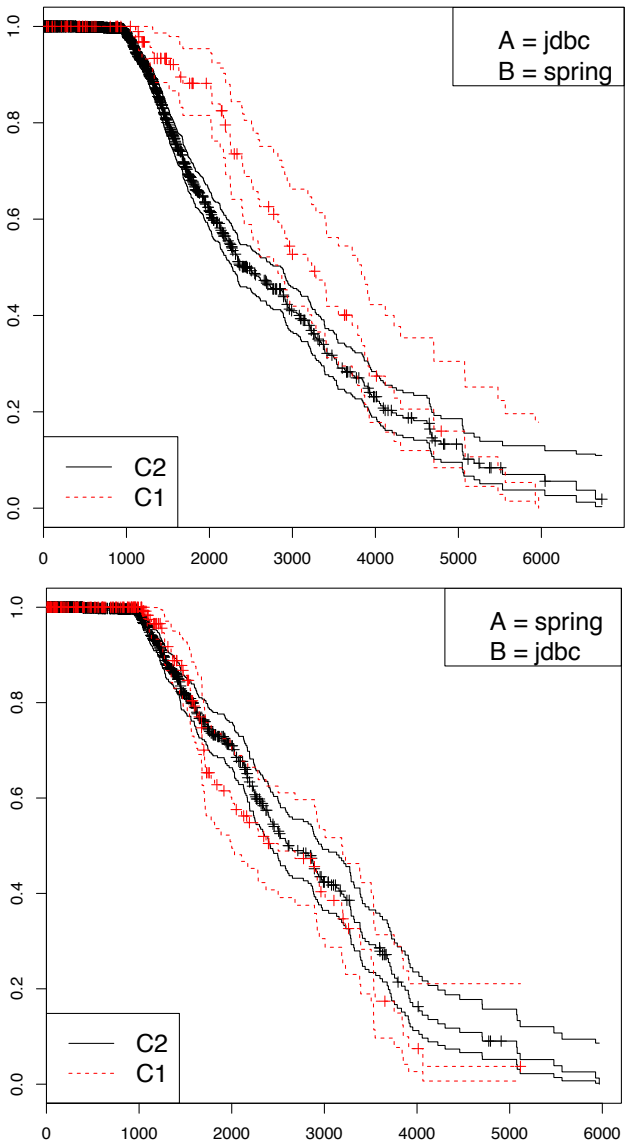

Fig. 4. Survival curves C_1 and C_2 for Spring and JDBC.

the survival probability of JDBC in the projects. Conversely, introducing JDBC when Spring is already present does not seem to affect the survival probability of Spring. We observed similar results for other pairs of frameworks. We did not observe any negative impact of the introduction of a framework on already present database frameworks.

We used Mantel-Haenszel tests to check for a difference in the survival rates, but did not find any significant evidence, most likely because the considered data sets were too small.

VI. Threats to Validity

Our results should not be generalized beyond Java projects, the considered database frameworks, or the used version control technologies. Our research also suffers from the same threats as other MSR research relying on Git and GitHub [15], [16].

By manually inspecting the names of the considered repositories, we observed that 470 of them appeared to be implicitly part of 117 projects. Clustering these projects into *logical* projects could provide more insight into the database framework usage analysis.

The detection of frameworks is based on the presence of specific import statements in Java files and specific XML-based configuration files in the project directory. This approach may have lead to false positives, since classes and interfaces made available with an import statement may be unused in the source code, and configuration files may be ignored while running the applications. A more detailed analysis of the source code could reveal if these components are actually used. However, the extra time required by such an analysis could be prohibitive for a large scale study.

During survival analysis, we assumed the probability for an event to occur to be the same for all studied projects. Some external factors may influence this probability, such as a change in the organisational policy of a framework or project.

VII. Conclusions

We studied the usage of five popular database frameworks in a large corpus of *GitHub* Java projects. We observed differences in survival rates, that did not seem to relate to the framework popularity. We observed an important co-occurrence, especially between JDBC and the other considered frameworks, but other combinations of database frameworks also seem to complement or reinforce one another. Such empirical evidence can be particularly useful to project developers desiring to introduce an additional framework, or to replace an existing framework by another one, as our analysis reveals which combinations and which replacements are more successful (in terms of survival) than others.

The research presented in this paper can be extended in many ways. We could consider projects in other forges or other programming languages. We could also include other Java database frameworks, and relate their survival to their popularity. We also aim to analyze framework survival at file level. Another extension includes mapping technologies for weakly structured databases such as NoSQL. Finally, traditional software metrics could be combined with more specific metrics reflecting the involvement of database frameworks in software projects in order to get a better understanding of the projects status, and particularly their maintainability.

Acknowledgment

This research is part of FRFC research project T.0022.13 financed by the F.R.S.-FNRS.

References

[1] J. P. Klein and M. L. Moeschberger, *Survival Analysis: Techniques for Censored and Truncated Data*. Springer, 2013.

[2] D. Kleinbaum, *Survival Analysis a Self Learning Text*, 2nd ed. Springer, 2005.

[3] P. M. E.L. Kaplan, "Nonparametric estimation for incomplete observations," *J. American Statistical Association*, vol. 53, no. 282, pp. 457–481, 1958.

[4] N. Mantel, "Evaluation of survival data and two new rank order statistics arising in its consideration," *Cancer Chemother Rep.*, vol. 50, no. 3, pp. 163–170, 1966.

[5] M. Goeminne, A. Decan, and T. Mens, "Co-evolving code-related and database-related changes in a data-intensive software system," in *CSMR-WCRE*, 2014, pp. 353–357.

[6] T. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. N. Nasser, and P. Flora, "Detecting performance anti-patterns for applications developed using object-relational mapping," in *Int'l Conf. Software Engineering*. ACM , 2014, pp. 1001–1012.

[7] A. Maule, W. Emmerich, and D. S. Rosenblum, "Impact analysis of database schema changes," in *Int'l Conf. Software Engineering*, 2008, pp. 451–460.

[8] D. Qiu, B. Li, and Z. Su, "An empirical analysis of the co-evolution of schema and code in database applications," in *Joint ESEC/FSE Conf.* ACM , 2013.

[9] I. Samoladas, L. Angelis, and I. Stamelos, "Survival analysis on the duration of open source projects," *Information & Software Technology*, vol. 52, no. 9, pp. 902–922, 2010.

[10] G. Scanniello, "Source code survival with the Kaplan Meier estimator," in *Int'l Conf. Software Maintenance*, 2011, pp. 524–527.

[11] P. Kyriakakis and A. Chatzigeorgiou, "Maintenance patterns of large-scale PHP web applications," in *Int'l Conf. Software Maintenance and Evolution*, 2014, pp. 381–390.

[12] M. Claes, T. Mens, R. Di Cosmo, and J. Vouillon, "A historical analysis of Debian package incompatibilities," in *Int'l Conf. Mining Software Repositories*. IEEE, 2015.

[13] M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," in *Int'l Conf. Mining Software Repositories*. IEEE, 2013, pp. 207–216.

[14] C. E. Bonferroni, "Teoria statistica delle classi e calcolo delle probabilità," *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, vol. 8, pp. 3–62, 1936.

[15] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. Germán, and P. T. Devanbu, "The promises and perils of mining Git," in *Int'l Conf. Mining Software Repositories*, 2009, pp. 1–10.

[16] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. Germán, and D. Damian, "The promises and perils of mining GitHub," in *Int'l Conf. Mining Software Repositories*, 2014, pp. 92–101.