Juggling with Data:
On the Lack of Database Monitoring in Long-Living Software Systems

Christian Zirkelbach
Software Engineering Group
Kiel University, Germany
czi@informatik.uni-kiel.de

Abstract

Long-living software systems often challenge associated software engineers and operators with changing requirements or increasing workload. In addition, performance issues or customer requests may cause inevitable software updates or refactoring tasks. These legacy systems are often based on outdated technologies and are poorly documented. In combination with insufficient knowledge of the (actual) system, the necessary transformations can be difficult. While the described situation is discussed in the software engineering community, databases are rarely covered. As databases are an essential part in almost every software system, they are affected by evolutionary tasks as well and have to face the same challenges. In this paper, we (i) describe problems based on non-existing database monitoring in long-living software systems and (ii) propose an approach as a solution, which addresses the described problems, and supports developers and operators in performing evolutionary tasks alike. Finally, we conclude and delineate open questions concerning our envisioned approach.

1 Introduction

During the life-cycle of a software system, software engineers and operators, who are responsible for the development and maintenance, have to face several challenges, as these systems evolve over time. An evolutionary step is often based on changing requirements and can be initiated by modernization aspects like cost-effectiveness or performance issues, which may be caused by inevitable software updates or increasing workload.

Software engineers, who maintain or extend software systems often have a lack of knowledge on these systems, which is needed to overcome these challenges. Similarly, operators require knowledge to perform various tasks, e.g., configuration, capacity planning, or performance monitoring and tuning, to maintain software systems. Long-living software systems are often part of a large software landscape and are based on obsolescent architectures and technologies, which tend to be poorly documented. In many cases this circumstance is worsened through insufficient knowledge of the legacy software system, which may be caused by, e.g., the age of a system, employed obsolete languages, technologies, and platforms, or retired colleagues [2].

In the area of software engineering, the described situation, regarding the evolution and refactoring of software systems, has been discussed in literature [4, 11]. Unfortunately, these descriptions rarely address the database, which is an essential part of or is at least employed in almost every software system, even if it is not distinguishable in the first place. Databases juggle with data – they provide well-defined interfaces for related applications based on standards, e.g., *JDBC* or *JPA*, store relevant application data, and allow further processing and analysis. The two latter tasks are often related to the terms *Data Warehousing* (technological) or *Big Data* (architectural), when talking about systems, which hold large amounts of data [9]. The key problem is the insufficient knowledge of (i) software systems and employed database systems and (ii) especially on the actual usage of these systems. For this reason, we propose a solution, which faces this problem by performing a live monitoring on software systems and related databases on the one hand and visualizing the results on the other hand. Finally, our approach is designed to aid developers and operators alike.

The remainder of this paper is organized as follows. In Section 2, we describe the problems based on the lack of database monitoring in software systems in long-living software systems. Afterwards, we present a first draft of our approach to counter the presented problems in Section 3. In Section 4, we discuss related work regarding our approach. Finally, the conclusions are drawn and open questions are delineated.

2 Problem Description

Godfrey and German [8] described that “software systems need documenters, who are also historians of the system. Their role includes documenting not only the system’s behavior, but also the evolution of the system from a more holistic point of view”. Without having such documenters, the previous described lack of knowledge is inevitable. Teams often have a *team*

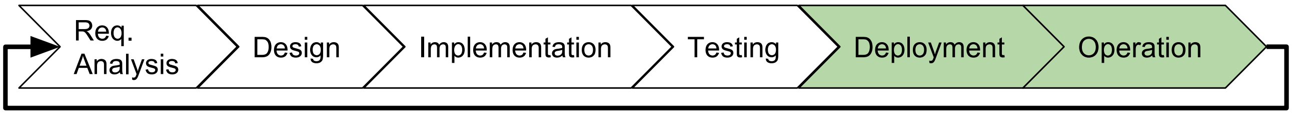

Figure 1: Extended system development life cycle (SDLC) based on [5]

historian, who is been asked, if a code related question occurs [6]. If this person leaves the team, a valuable and crucial amount of knowledge is lost. This absent information is essential for accomplishing evolution and adaption on the development side, but it is also important for the daily business in the operating as well.

In Figure 1 an extended version of the system development life cycle (SDLC) is shown. The SDLC describes an approach from planning towards developing an information system [13]. In our version, the SDLC consists of six, consecutive phases from *Requirement Analysis* to *Operation*. Every software system passes, at least one time, through the entire cycle. The *Evolution* phase, as mentioned in Section 1, spans the first four illustrated phases (colored in white). In order to successfully perform an evolutionary task on a software system, an extensive amount of information is needed. This information needs to be collected and evaluated at least in the *Deployment* and *Operation* (colored in green) phases to be an useful input for the proposed evolution. Especially in the context of long living software systems, the documentation of employed or embedded databases is often insufficient. Changes towards the database management system, or included data structures like schemes, tables, and queries, affect directly the usage of the database, but the related software system as well. If these changes are undocumented and cause problems, the software system can become unstable or unreliable. Furthermore, as often operators (database administrators) are responsible for the latter changes, there may be a communication gap between operators and developers, which further complicates this process.

Based on our own professional experience in combination with extensive discussions with operators and developers in industry, there is a need for appropriate monitoring and visualization on the usage of databases in (long-living) software systems. Additionally, we conducted a performance analysis on the long-living open repository software EPrints, which is been developed for more than fifteen years [14]. Our results showed, that database-related bottlenecks existed within the software. These findings led to a performance improvement in the following released version. This work substantiates the need for appropriate database monitoring in long-living software systems.

To the best of our knowledge, there exists no suitable approach, which faces the described problems and especially addresses developers and operators. For this reason, we propose an approach as a solution to support system and database comprehension in large software landscapes. We focus on the evolution and actual usage of software systems and related databases.

3 Approach

Our envisioned approach includes four consecutive activities (A1 to A4), which are briefly described in the following. Figure 2 illustrates an overview of the activities in our approach.

A1 – Monitoring: Within a software landscape existing applications and included or related databases are *monitored*. The gathered information, basically executed database related methods, will be provided in form of a data stream, which contains monitoring records.

A2 – Analysis: Based on the data stream, we *analyze* and process the monitoring records. The processing contains – (i) the reconstruction of monitoring records into corresponding traces and (ii) the aggregation of similar traces. Furthermore, we create a persistent data model for the software landscape, which is needed as we want to enable a live visualization.

A3 – Transformation: To enable a visualization of our database operations and related databases, we need to *transform* the data model into a visualization model.

A4 – Navigation: We offer three different visualization perspectives. The user can navigate between them. The first visualization features a landscape-level perspective, orientated on UML deployment diagrams, which provides a good overview of the reconstructed software landscape based on the monitored applications and databases. In comparison, the architecture-level perspective offers a database specific visualization based on the entity relationship model (ERD) [1], which is a common used visualization of relational databases. The visualization shows employed database tables and relations between them. Possibles use cases are providing an overview of the database usage for development and operation. The last perspective, presents a live usage-level visualization, which is based on the 3D city metaphor [7]. The visualization shows the actual database usage of a single application or an entire database. Possibles scenarios are exploration and enhanced system and database comprehension.

For performance analysis purposes, we plan to provide an additional visualization, which will be inte-

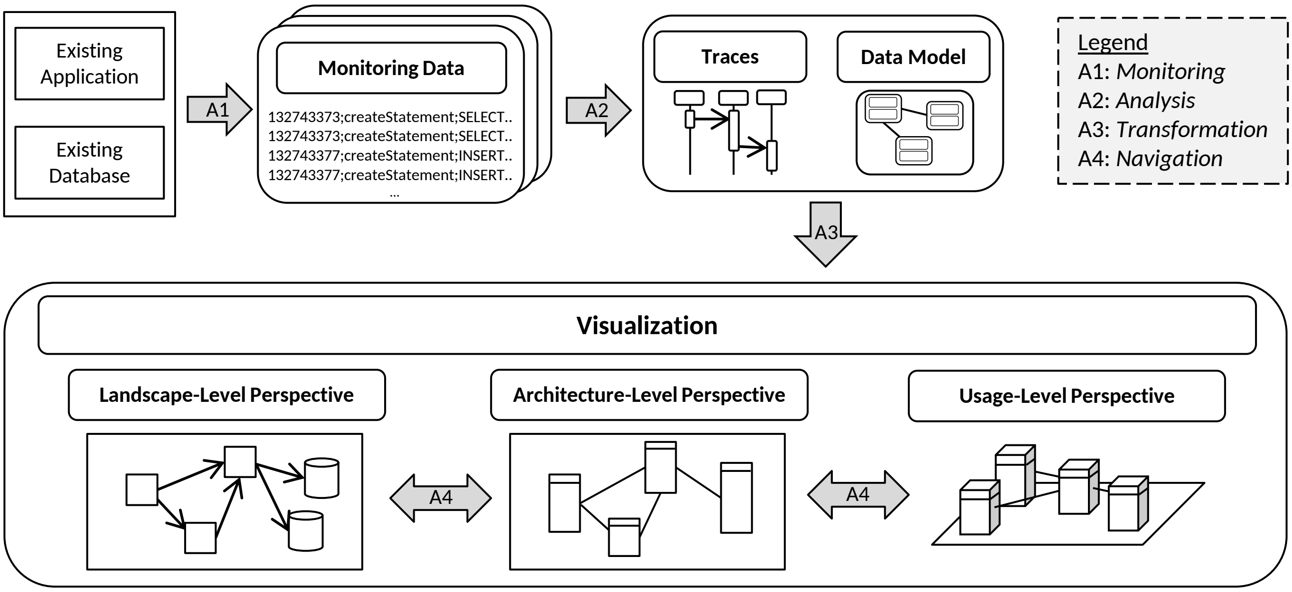
Existing Application
Existing Database
A1
Monitoring Data
132743373;createStatement;SELECT..
132743373;createStatement;SELECT..
132743377;createStatement;INSERT..
132743377;createStatement;INSERT..
...
A2
Traces
Data Model
Legend
A1: Monitoring
A2: Analysis
A3: Transformation
A4: Navigation
A3
Visualization
Landscape-Level Perspective
Architecture-Level Perspective
Usage-Level Perspective
A4
A4

Figure 2: Overview of our envisioned approach

grated into the three presented perspectives, based on the representation of call tree views (CTV) [3]. Employing CTVs allows to represent the hierarchy of a database call and simplifies filtering and sorting tasks. This feature is particularly interesting for operators, when conducting database optimizations, finding performance issues [15], or evaluating database performance [10]. As our envisioned approach is only a first draft, a more detailed description, particular in respect of the visualization, will be published in near future.

4 Related Work

There are several approaches, which are related towards our envisioned approach, based on the methodology, visualization, or research topic. Due to space restrictions, we only list closely related work focusing on the visualization.

ExplorViz [16] is a web-based monitoring and visualization tool for large software landscapes. Furthermore, it features two distinct visualizations, showing a monitored software landscape on the one hand, and an application level visualization on the other hand. In contrast, our approach focuses on the monitoring and visualization of software systems and related database systems, in order to facilitate system and database comprehension.

DAHLIA [12] provides visual analysis of database schema evolution. The tool harvests the history of database schemes from a software repository based on static analysis and features an interactive 3D visualization for exploring software evolution. In their recently released version 2.0, they included support for Object-Relational-Mapping (ORM) frameworks [17]. This allows to analyze the evolution of a database over its lifetime more precisely. In contrast to DAHLIA, our approach utilizes dynamic analysis to obtain a live visualization of the database and executed database queries from associated software systems.

InspectIT[1] is an open source application performance management (APM) tool, which provides a monitoring sensor for databases. Based on the gathered information, it is possible to perform a performance analysis on executed database queries. Although, analyzing the performance of executed database queries is an important use case within our approach, we draw our primary attention on the visualization and the previously presented perspectives, in order to aid the comprehension process.

5 Conclusions

In this paper, we reported on the lack of database monitoring in the context of long-living software systems. Afterwards, we described problems based on the absence of database monitoring and revealed the need for detailed documentation and appropriate tooling in this context. Furthermore, we proposed a first draft of our approach as a solution, which faces the discussed challenges, and aids developers and operators alike. Our approach facilitates the monitoring of applications and related databases and provides basically three different visualization options.

First, we offer a landscape-level perspective, orientated on UML deployment diagrams, which provides an overview of the monitored applications and related databases. Second, we provide an architecture-level perspective, which is inspired by the ERD, and reveals employed database tables and relations between them. Finally, we present an usage-level perspective, based on a 3D city metaphor, which allows the user to take a look on the actual usage of database by

[1] `https://github.com/inspectIT/inspectIT`

a specific application or an entire database at once. All three visualizations have in common, that they provide a visual abstraction of the displayed objects. Only necessary information is shown in the first place and further data and visualization objects are revealed on demand, in order to avoid overstraining the user. Our open questions are:

- Which layout is suitable for our landscape-level perspective, that comprises the complete software landscape including the databases?
- How do we link databases and related artifacts (e.g. deployed Software) for our visualization?
- Which concrete visualization is suitable for different developer and operator scenarios in the usage-level perspective?
- Does our 3D visualization within our usage-level perspective offer an advantage over a traditional 2D visualization like the architecture-level perspective?
- Are our three perspectives in combination with further monitoring information appropriate for performing a performance analysis?
- How can we successfully combine our database monitoring and visualization approach with an APM tool?
- Which related approaches or tools could be employed, when evaluating our approach within a controlled experiment?

References

[1] P. P.-S. Chen. “The Entity-Relationship Model – Toward a Unified View of Data.” In: *ACM Trans. Database Syst.* 1.1 (Mar. 1976), pp. 9–36.

[2] V. Raijlich et al. “Software cultures and evolution.” In: *Computer* 34.9 (Sept. 2001), pp. 24–28.

[3] W. De Pauw et al. “Visualizing the Execution of Java Programs.” In: *Software Visualization.* Springer, 2002, pp. 151–162.

[4] T. Mens and T. Tourwé. “A Survey of Software Refactoring.” In: *IEEE Trans. Softw. Eng.* 30.2 (Feb. 2004), pp. 126–139.

[5] D. Avison and G. Fitzgerald. *Information Systems Development: Methodologies, Techniques and Tools.* 4th. Information systems series. McGraw-Hill Higher Education, 2006.

[6] T. D. LaToza, G. Venolia, and R. DeLine. “Maintaining Mental Models: A Study of Developer Work Habits.” In: *Proceedings of the 28th International Conference on Software Engineering.* ICSE ’06. Shanghai, China: ACM, 2006, pp. 492–501.

[7] R. Wettel and M. Lanza. “Visualizing Software Systems as Cities.” In: *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis.* 2007, pp. 92–99.

[8] M. Godfrey and D. German. “The past, present, and future of software evolution.” In: *Frontiers of Software Maintenance, 2008. FoSM 2008.* Sept. 2008, pp. 129–138.

[9] A. Cuzzocrea, I.-Y. Song, and K. C. Davis. “Analytics over Large-scale Multidimensional Data: The Big Data Revolution!” In: *Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP.* 2011, pp. 101–104.

[10] S. Ray, B. Simion, and A. D. Brown. “Jackpine: A benchmark to evaluate spatial database performance.” In: *Proceedings of the 27th International Conference on Data Engineering.* Apr. 2011, pp. 1139–1150.

[11] Z. Durdik et al. “Sustainability guidelines for long-living software systems.” In: *Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM).* 2012, pp. 517–526.

[12] L. Meurice and A. Cleve. “DAHLIA: A visual analyzer of database schema evolution.” In: *Proceedings of the IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE).* 2014, pp. 464–468.

[13] J. S. Valacich, J. F. George, and J. A. Hoffer. *Essentials of Systems Analysis and Design.* 6th. Pearson Education, 2015.

[14] C. Zirkelbach, W. Hasselbring, and L. Carr. “Combining Kieker with Gephi for Performance Analysis and Interactive Trace Visualization.” In: *Symposium on Software Performance 2015: Joint Developer and Community Meeting of Descartes/Kieker/Palladio.* 2015.

[15] T. H. Chen et al. “Finding and Evaluating the Performance Impact of Redundant Data Access for Applications that are Developed Using Object-Relational Mapping Frameworks.” In: *IEEE Transactions on Software Engineering* 42.12 (Dec. 2016), pp. 1148–1161.

[16] F. Fittkau, A. Krause, and W. Hasselbring. “Software landscape and application visualization for system comprehension with ExplorViz.” In: *Information and Software Technology* (2016). http://dx.doi.org/10.1016/j.infsof.2016.07.004.

[17] L. Meurice and A. Cleve. “DAHLIA 2.0: A Visual Analyzer of Database Usage in Dynamic and Heterogeneous Systems.” In: *Proceedings of the IEEE Working Conference on Software Visualization (VISSOFT).* 2016, pp. 76–80.