DevOps Continuous Integration: Moving Germany's Federal Employment Agency Test System into Embedded In-Memory Technology

Eldar Sultanow
Capgemini, Germany
eldar.sultanow@capgemini.com

Alina Chircu
Bentley University, USA
achircu@bentley.edu

Andreas Jung
Capgemini, Germany
andreas.jung@capgemini.com

Martin Blomenhofer
Federal Employment Agency, Germany
martin.blomenhofer@arbeitsagentur.de

Philippe Masson
Capgemini, Germany
philippe.a.masson@capgemini.com

Abstract

This paper describes the development of a continuous integration database test architecture for a highly important and large software application in the public sector in Germany. We apply action design research and draw from two emerging areas of research – DevOps continuous integration practices and in-memory database development – to define the problem, design, build and implement the solution, analyze challenges encountered, and make adjustments. The result is the transformation of a large test environment originally based on Oracle databases into a flexible and fast embedded in-memory architecture. The main challenges involved overcoming the differences between the SQL specifications supported by the development and production systems and optimizing the test runtime performance. The paper contributes to theory and practice by presenting one of the first studies showing a real-world implementation of a successful database test architecture that enables continuous integration, and identifying technical design principles for database test architectures in general.

1. Introduction

DevOps is an emerging software engineering paradigm in which software development and operations teams work together, rather than in silos, to build, test, and release software to users, as well as to monitor the behavior of the software in use and plan improvements [12, 14, 15]. DevOps incorporates agile principles through continuous planning, integration, deployment, testing, and monitoring, as well as through communication, collaboration and automation [14, 15, 29, 30, 33]. A core practice in DevOps is continuous integration (CI), where code is checked frequently into a central repository and automated builds and tests are run. This reduces time lost with manual tests, detects errors quicker, improves software quality, and reduces update time [10, 15, 30, 33]. DevOps practices could increase software process innovation [28] as well as the performance of entire information technology (IT) departments and companies [10], enabling them to quickly react to market needs [33].

While DevOps is starting to be adopted in a variety of IT projects, its adoption for database-focused projects is lagging [7], despite earlier predictions [1]. In addition, as the reader will shortly see, few academic studies on how DevOps practices can be used for database development exist. This paper attempts to shed light on this emerging area of both practical and academic importance – applying DevOps CI practices to database development.

Our focus is a very large, important project in the public sector – the development of an innovative test architecture for the online transaction processing (OLTP) system at Germany's Federal Employment Agency. The agency is focused on workforce placement and training programs for individuals and firms, unemployment and child benefits, and labor market research and reporting. It works with a network of branch offices and local agencies and strives to provide services in a timely and efficient manner. Its OLTP system, one of the largest federal OLTP applications in Germany, supports one hundred-thousand transactions each day and processes over EUR 25 billion per year. With the increasing complexity and the rapidly changing requirements of highly specific functionalities, the volume and complexity of tests are steadily increasing Tests need to be performed frequently and have fast setup and runtimes. As system quality is very important, they cannot be skipped or simplified. The increased testing time can slow down changes and decrease the service timeliness and efficiency, which are the cornerstone of the agency's

URI: https://hdl.handle.net/10125/60185
ISBN: 978-0-9981331-2-6

mission. In the next sections, we review relevant prior work, detail our methodology, explain how we designed, built and evaluated a novel test architecture to solve this problem, and discuss challenges, contributions and limitations.

2. Related work

Several recent systematic literature reviews of DevOps highlight the definitions of this emerging concept and its practices, benefits and challenges [12, 14, 28]. The number of relevant articles identified in these reviews is relatively small (49-60 from technical databases, such as IEEE and ACM, and only 6 in the information systems AIS library) [12, 14, 28]. Thus, it seems that DevOps research is still in its early stages, especially in specific disciplines, such as information systems [28] or in specific application areas, such as embedded systems [19].

None of the existing review articles identify specific DevOps applications to database development. To identify research on how DevOps, in general, and CI, in particular, are used in database development, we conducted a systematic literature review [17]. We searched for "DevOps" and "database" or "continuous integration" and "database" in the abstract, title and keywords (if available) of peer-reviewed articles from several major databases - IEEE Xplore, ACM Digital Library, Elsevier ScienceDirect, EBSCO Academic Search Complete and Business Source Complete, and AIS Electronic Library, and found a total of 32 articles (IEEE – 16, ACM – 10, Elsevier – 6). After elimination of 3 duplicates, the article metadata was analyzed for relevance, and 21 articles were further eliminated because they did not specifically address DevOps or CI in database development or were editorials or special issue introductions. The final list of 8 articles is included in Table 1.

Most articles appeared in ACM and IEEE publications (3 each); only 2 articles appeared in Elsevier publications. This suggests most research on the investigated topic is very technical. The majority of articles appeared in the last four years (5), and there are multi-year intervals among the publication years of the remaining articles. Only 3 articles appeared in journals (the rest are conference proceedings and keynote speeches). Most articles address the development of a specific application, such as databases for physics experiments, software collaboration, mashup interfaces, high performance systems, etc., and describe using DevOps or CI principles, but the corresponding DevOps or CI implementation details are rarely presented. A few articles discuss more general principles for DevOps automation or test-driven development, but do not provide many technical details. Taken together, this review confirms that few peer-reviewed studies on how DevOps and CI practices can be applied to database development in a real-world environment exist, thus suggesting that our research topic is timely and useful.

3. Methodology

The methodology adopted in this paper draws from the action research method, as applied to the information systems field [3, 4]. Action research is useful for solving practical problems and creating organizational change,

Table 1. Systematic literature review summary

Reference	Year	Source	Focus	Contributions
[1]	2007	IEEE	Test-driven database development	Review of database-focused test-driven methods: regression testing, refactoring, and continuous integration methods
[18]	2009	IEEE	Implementation of CI at Launchpad	Lessons: don't block on fears, try it out; make it easy to run integration tests; a continuous integration system is a big enabler
[8]	2013	Elsevier	Database for JET (European project for plasma physics experiments)	System design (and application of CI for JET database improvement)
[11]	2015	ACM	DevOps challenges, benefits and best practices (conference keynote)	Milestones: feedback loops between Dev and Ops, automated performance monitoring, key performance metrics in CI, Test and Ops, tools and performance metrics sharing across teams
[5]	2015	ACM	Development of a high performance computing system	System design (based on DevOps principles)
[25]	2016	ACM	Implementation of CI in a specific database - SAP HANA	Methods: performance benchmarks for pre-commit tests, exceptions, and restricting the amount of manual work for CI to a minimum
[9]	2016	Elsevier	Interaction data acquisition system for mashups	System design (and application of CI practices for integration of the system into mashup applications)
[32]	2017	IEEE	Removing borders between development and operations (conference keynote)	Methods: analyzing log data for deployment and integration monitoring, test input generation for reproducing crashes and testing complex queries, zero downtime schema evolution and deployment

while also studying the process and learning from it [4]. The method involves a diagnostic stage (problem definition and hypothesis formulation) and a therapeutic stage (change introduction and evaluation). To increase rigor, the method is structured in five phases, which can be repeated as needed: "(1) diagnosing, (2) action planning, (3) action taking, (4) evaluating and (5) specifying learning" [3]. Key features of action research - iterative process, theory-practice links, collaboration between researchers and practitioners, and learning by both practitioners and researchers [4; 26] - make it different from mere consulting [3]. The results of action research include "double-loop learning" for the organization (which can use learnings to restructure its norms), diagnosing problems for future work (if the current attempt was unsuccessful in solving the problem), and feedback on the applicability of theoretical frameworks (which can help improve existing frameworks or develop new ones) [3].

Because our project focuses on the development of an IT artifact, we also draw from design science research. Design science research, also called design research, is a rigorous process for building IT artifacts, evaluating their usefulness, and developing prescriptive design knowledge, also called design principles [26, 31]. It is usually organized as a stage-gate process: developing awareness of a problem (from industry knowledge or literature), creating solution suggestions, developing and implementing the artifact, evaluating it, and concluding the effort by writing and communicating the results [26, 31].

A combination of these two research methods has recently been proposed under the name action design research (ADR), defined as "a research method for generating prescriptive design knowledge through building and evaluating ensemble IT artifacts in an organizational setting" [26]. ADR recognizes the importance of the organizational context and of the interests, values, and assumptions of various stakeholders in shaping the design and development of IT artifacts. ADR is organized in several phases guided by principles: (1) problem formulation (guided by practice-inspired research and theory-ingrained artifact), (2) building, intervention and evaluation (guided by reciprocal shaping between the IT artifact and the organizational context, mutually influential roles of researchers and practitioners, and authentic and concurrent evaluation of the IT artifact emphasizing authenticity over a controlled setting), (3) reflection and learning (guided by the emergence of the IT artifact from actions taken in phases 1 and 2), and (4) formalization of learning (guided by generalized outcomes, which could include generalization of the problem instance, generalization of the solution instance, and development of general design principles). Iterations of phases 1 and 2 can be conducted as needed, phase 3 reflection and learning can take place after each one of the preceding stages, while phase 4 happens at the end of the research process [26]. ADR is starting to be applied to a variety of problems, such as developing health and wellbeing platforms in the context of living labs [16]. In this paper, we recognize the above-mentioned benefits of each individual method - action research and design research – but also the importance of the context in shaping the artifact during development and use, and therefore use ADR as an overarching methodology.

4. Continuous integration at Germany's Federal Employment Agency

Next, we describe the problem, our design and implementation decisions, the resulting challenges, and the iterations made to shape the resulting solution. In this process, as suggested by ADR [26], we use industry best practices and researcher technical expertise (obtained from extensive consulting knowledge and IT work experience), existing research (reviewed previously and in the next sections as needed), and the studied organization's own context and technical environment. We also highlight emerging design principles which could be generalizable, as discussed further in the conclusions section.

4.1. The existing test environment and its shortcomings

The production environment for the application we study in this paper includes a database with a current size of 45 TB, which is increasing by about 12 TB per year. This database is deployed in an Oracle Real Application Clusters (RAC) system, in which two databases of this size are coupled together in one cluster to ensure failure safety, high availability and performance, and scalability, among others. The programming language for the application is Java.

Before the start of the project, the development and test environment and the production environment were designed to be compatible, based on a shared full-license instance of Oracle. This database server that ran on a powerful system was shared by all developers. The advantage was maximum compatibility with the production database, as the software version included all features available in the production environment. The main disadvantages were high input/output (I/O) wait times (network and hard disk) and high usage of hard disk space when developers and build systems processed large amounts of data. In addition, the

database schemas gradually diverged in their setup (including permission changes and individual procedures) and an initialization time of up to 15 minutes was required for tests. Initially, this setup time was fast enough, but the increasing number of tests resulted in much slower test runtimes. When we started the project, the main problem was that database system failures or network problems paralyzed the entire development, resulting in unacceptable delays.

4.2. Deriving design principles for the test environment

The first factor is the **non-feasibility of comparability between the development (and test) and production environments**. Such a configuration is not efficient for unit testing purposes, as the organization's own experience indicates. In addition, existing research suggests that waiving development-production comparability for testing purposes is a reliable option (for example, when HSQLDB is used as the local development database even though the production database is Oracle) [23].

The second factor is the **requirement for continuous integration**, which, as explained earlier, enables each developer to verify code changes easily and quickly from a test-build and confirm that there are no unforeseen side effects. If tests take hours or days, the developer will start dealing with other issues and will have to re-familiarize with the test details when the results are ready. As each developer works locally in his own development environment, each workstation needs to have its own database system to facilitate CI.

The final factor is **minimizing license costs**. Large projects can involve tens to hundreds of developers (in our case, around 70 to 80), each requiring a full database license, which can be very costly. The free versions of mainstream databases such as Oracle XE or Microsoft SQL Server Express are not usable since they have significant features and hardware utilization limitations (limited amount of persistent hard disc space and/or memory and/or CPU cores).

These factors justify the use of a development and test database system that differs from the one in the production environment. Homogeneity of the databases used in the development and production environments in the organization studied in this paper is already a moot point, since, as explained previously, the data volumes and infrastructure configuration (RAC vs. no cluster) as well as the range of features for development and production are already different. This renders any attempts at homogeneity pointless.

This analysis led to a quick transition to a decentralized approach, using a local Oracle XE instance for every developer. Since the central instance differed greatly in terms of performance, database size and features compared to the production environment, testing performance or runtime behavior generated meaningful results. This change also had other benefits: Network I/O wait times were reduced, since requests were routed over the local loopback device (a virtual network interface). Hard disk I/O wait times also decreased, given that fast data access was guaranteed locally on a solid-state drive (SSD).

However, the runtime of the tests was still high - up to 55 minutes (30 minutes for the tests, 15 minutes to initialize the database and 10 minutes to build the application). This was significantly higher than the recommended maximum total runtime for a build (including tests) of 15 minutes (to enable developers to prepare for the review or take a small break, then continue working on the same issue and fix build/test errors without switching to another task). This suggested an alternative database system was needed.

4.3. Deriving design principles for an alternative database system

A **throw-away database** - a lightweight database which can be quickly set up when a test needs to be executed – is best. This guarantees the same database configuration across all development instances and for all pushes into the source code repository, ensures direct comparability of test results, and removes the need for long-term troubleshooting due to different behavior of differently configured installations.

The database must **run on different platforms and with multiple instances on the same machine**. This is needed because the server environment is based on a Linux host with a middleware image that cannot be changed by the project, but the developer machines are Windows-based. Additionally, the project uses many virtual environments (Java Virtual Machines, or JVMs) on a real server host to minimize hardware costs.

The database system should **support the SQL standards** as much as possible. Since the application has been in use since 2014 and is represented via a host of initialization scripts, the migration overhead onto a new database system should require minimal adjustments. We do not want two distinctive initialization scripts for two systems, but rather want to reach 100% equality in order to minimize errors.

The database system should **support the language extensions of the production system** (in this case, Oracle). Since the application has long been in use, the initialization scripts and some native queries in the application contain SQL code that is manufacturer-

specific and which we should be able to reuse in the new database system when possible.

The database needs a **good performance to ensure low test-build runtimes and should be suitable for unit tests** (i.e. it needs to be designed to handle many small transactions). I/O wait times (network wait times, hard disk access) need to be minimized.

Finally, the database system needs to be **portable in terms of distribution and the number of running instances**, as a total test time of at most 15 minutes is not possible by simply having a fast database. Integration tests require a significant amount of calculation time within the application server itself, which is unaffected by the database performance.

The above-mentioned requirements are congruent with the need for a **lightweight, portable, embedded, in-memory database management system** [22, 24, 27], which we decided to use for this project instead of the disk-based database systems tried in previous iterations. In-memory data management enables companies to derive competitive advantage by analyzing huge amounts of operational business data in real-time to investigate new scenarios and business opportunities [24]. In-memory data management, supported by recent hardware advances such as multi-core architectures and larger and cheaper main memory, performs better than current relational database management systems (RDBMSs) (which can speed some computations by caching frequently used data in memory but still require disk access to perform most queries) [24, 35]. Eliminating the disk input/output bottleneck creates efficiencies in indexing, data layout, parallelism, concurrency control and transaction management, query processing, fault tolerance, and data overflow [35].

In-memory databases have been successfully tested in a variety of applications. They can help develop real-time supply chain analytics decision systems [13], enable faster data processing, more flexible data access, more up-to-date data, less aggregates and deeper drill-down capabilities [2], reduce latency times and enable processing and analysis of large amounts of data [20, 34] with many concurrent users [20]. Use cases showing the benefits of in-memory databases for emerging DevOps practices such as continuous integration are also starting to emerge [25]. The move to in-memory databases affects the hardware choice, the software architecture and the software development paradigms [24]. In our case, the test architecture and the testing paradigm are affected, as explained next.

5. Performing the change

Although the change to a local Oracle XE database instance was helpful, some disadvantages remained. Compared with the full license version of Oracle, the feature support was restricted, network and hard disk I/O wait times were still present, parallel test execution was still not easily possible and database schemas, as well as basic settings of instances, continued to diverge. The runtime decreased slightly but worsened over time as the number of test cases increased.

The next step, suggested by our analysis of an alternative database system, was to switch to an embedded in-memory database with Oracle language support. Using an embedded database removes any network I/O wait times completely. This is because no data serialization is required for network calls, as no system call is needed to send data over a network interface. Likewise, disk I/O wait times are removed, since the entire database is loaded and accessed solely in memory. Database schemas, as well as the basic settings of the database, are guaranteed to be the same

Table 2. Comparison of different database systems

	SQL Server	Oracle	IBM DB/2	MySQL	PostgreSQL	Derby	HSQLDB	H2
Type of installation	Preinstalled			(Partially) preinstalled		On-demand		
Portable	-			✓[1]		✓	✓	✓
SQL standards	✓					✓[2]	✓	
Supports Oracle's language features	-	✓	-				✓[3]	
In-memory support	✓[4] (Tables only)			✓ (Tables only)		✓ (Full DB and table-specific)		
Runtime-embedded	-					✓		
Overheads	Networking, disk writes[5]			Networking, disk writes[5]		None		
Supports native SQL functions/procedures, triggers	✓	✓	✓	✓	✓	-	DML only	-

Notes: (1) Binaries can be started without prior installation, but initial setup via properties is required (user, schema, rights) (2) Support is restricted up to SQL Standard 99 (3) Support is restricted in terms of accepted data types, clauses, automatic data conversion, and conversion of return values (4) SQL Server 2014 / Oracle Database 12c / IBM DB2 10.5 is required; the in-memory table feature requires appropriate license fees (5) Disk writes overhead can be reduced by the use of in-memory tables

for each developer and each test run, as the database is initialized on-demand in the same way on every machine. In addition to a fast runtime, the database initialization can be reduced to just seconds by using dumps (directly on file level/directly provided in JAR files). Parallel test execution is possible as long as there is enough memory to start up multiple databases and the test cases are the only data in the database. One disadvantage remains, however, which is that the feature and language support is limited compared to the full-featured enterprise version of Oracle.

5.1. Moving to an in-memory database approach

To select the best database, we performed an evaluation of the various options (with partial or full in-memory support) (see Table 2). Based on the above requirements and evaluation results, HSQLDB and H2 are leading the feature list and can be considered for the desired change. HSQLDB is a lightweight and high-performance database that supports high concurrency [6, 21]. Although H2 is often regarded as a successor to HSQLDB, both database systems have evolved further since their branching. Three core advantages of HSQLDB over H2 were identified.

1. H2 does not fully support the SQL:2003 standard command MERGE INTO (or does so in a different way), whereas HSQLDB does. In our case this command is widely used in initialization scripts and the application server and the conversion would be costly.
2. HSQLDB is the only in-memory database that supports SQL-based procedures and functions. Although there are not many of these in the application studied here, the few that are present are essential and would entail a noticeable conversion effort. In HSQLDB, only the syntax needs to be adapted and some refactoring to the SQL standard is required. All special procedures with manufacturer-specific statements are used for partitioning purposes, which neither work in the Oracle XE instance nor are necessary for tests.
3. H2 does not support schema cleaning via the SQL command TRUNCATE SCHEMA, which is a valuable feature. This statement would have to be split up into individual TRUNCATE TABLE statements, which require more time to execute and result in longer test run times.

Another benefit was that the database could be used for all test/analysis cases not requiring special database features and not testing database performance. Examples include data throughput and load behavior of the application servers or interface calls, as well as automated user interface tests. Also, given that refactoring (restructuring of existing code without altering its behavior) of manufacturer-specific or unprocessed SQL queries to the SQL standard was performed for the in-memory approach, compliance with standards was increased and the error potential by custom PL/SQL code was reduced. Finally, by making the database system exchangeable, complex error cases (when both the database and the application are faulty) could be analyzed more easily, as developers could switch the database and remove the faulty behavior or debug the behavior interactions of both systems by debugging the database as well.

5.2. Deriving design principles for implementation

To ensure a successful move to HSQLDB, several principles were identified as follows: amendments to the application, adaptation of the database initialization and migration scripts, and run-time optimization.

The **amendments to the application** comprise the need to change the DataTypeFactory for DB-Unit (trivial), data type conversion problems (casting) due to different handling of return values by Oracle and HSQLDB JDBC (a Java application programming interface for client database access) drivers, and the fact that native queries with extended SQL standard or special Oracle-specific commands were no longer executable. The latter problem was solved by refactoring to avoid using native queries and by making all statements comply with JDBC/JTA (Java application programming interface) standards. If this was impossible, different queries were implemented (which could lead to higher maintenance efforts).

We also had to adjust properties (for example language, time and region settings) and refactor the database Persistence Manager to use different database drivers and credentials for setting up connections and running queries, so that both databases are supported by the application. However the number of database connections established over the runtime of a full test-build quadrupled from 5 to over 20. This issue is still being analyzed. As this number is still manageable, the limit of simultaneous connections was set to 32.

The maximum memory usage of the test suite was expanded from 4 to 6 GB. If the Java Garbage Collector is used to free up memory, 5.5 GB are enough. The database needs up to 1.5 GB of additional memory, which is mainly required for BLOBs (Binary Large Objects, usually multimedia files).

Adaptation of the database initialization and migration scripts was necessary, since Oracle-specific commands did not work. This included session

configuration for settings such as language, date and time format, length semantic etc. For the tests, the default settings were sufficient.

Partitioning is generally not available in HSQLDB and was not used in Oracle XE previously (since it is not supported in the free version). Partitioning is only required for large data sets, which are not used in test-builds. Additionally, physical storage settings proved unimportant for the test system and were restricted to be executed only on Oracle.

The procedures had to be adjusted syntactically, were refactored to match the SQL standard or were transferred into the applications code. Anonymous PL/SQL statements were reduced to standard statements or moved into temporary procedures.

The implicit creation of indices for foreign keys (FK) ("… USING INDEX") was not performed by default in HSQLDB, so any FK creation was split up in index and FK creation statements. This is an industry best practice step, as Oracle has different behavior for system-created indices versus user-created indices, which can lead to complexities during error analysis.

Oracle supports constraint creation with the modifier "ON DELETE SET NULL", even for columns that are defined as not-null. These constraint creation statements were refactored to SQL standard.

Some data types failed in HSQLDB – namely float data types with a scaling of 126, which is too high for business scenarios. The value was therefore reduced to the default scaling. Additionally, the maximum sequence value was extremely high and exceeded the maximum value for the Java long data type. As this was not required by business scenarios, it was reduced to the maximum value for the Java long data type.

Two keywords were not supported in HSQLDB. These were the keyword "VIRTUAL" and the index creations with "… PARALLEL [NUMBER]". The first keyword was removed and the required behavior for automatically generated column values was achieved with the "… GENERATED ALWAYS AS …" statement modifier that was still supported. The latter keyword has no equivalent in HSQLDB and was removed without being replaced (without causing problems, as test cases only use a small amount of data sets and were practically unaffected).

For **run-time optimization,** we pursued several measures. Since the largest test runtime is not required for database access, but only for calculations in the application server, the reduction from approximately 1 hour runtime to a maximum of 15 minutes cannot be achieved by merely using an in-memory database. It would be necessary to reduce the number of tests or distribute the test load. Junit (Maven Surefire plugin) supports parallel test execution by forking, but to take advantage of this, a clean database is required for each test. Otherwise, the tests may influence each other due to concurrent changes in the database tables. Also, some tests leave data behind and can only be executed failure-free when they are processed sequentially and in a specific order. The database should therefore be re-initialized for each test. However, even if the database re-initialization has been reduced by the in-memory approach from 15 minutes to a few seconds (loading a database from an initialized dump takes less than 2s), when there are thousands of tests, restarting the database significantly increases the runtime. Hence, the database should be cleaned up between tests without restarting. There are two options for this.

1. Each test cleans up the changes it makes in the database. The problem here is that the tests would have to record any changes made in the database to clean them up afterwards. If, however, the business logic of the application changes, then more may be written to the database than is assumed by the test. To find any such cases, all the tests would have to be checked for compatibility with every schema change, resulting in exploding costs.
2. The removal of all written data ensures that the database is in its initial state before running a test. This would be performed by deleting all the data without changing the current database structure. To do this, the following statement is executed in HSQLDB between each test run: TRUNCATE SCHEMA [SCHEMA_NAME] RESTART IDENTITY AND COMMIT NO CHECK.

We chose the second option. The runtime performance was high due to deactivation of constraint-checks (less than or equal to 1ms with just the in-memory model without persistence).

Another problem is that some master data sets entered in the database during initialization (constants, calculation rules, time frame settings and BLOBs) are required for the application's business logic. One solution is to re-initialize only relevant tables affected by a test run between each test. However, the runtime of these SQL statements is still longer than 100ms due to table sizes (some tables have tens of thousands of rows and some have BLOBs).

Another solution is to exclude the tables containing these master data sets from truncation. This requires a "TRUNCATE TABLE [TABLE_NAME]" statement for each individual table instead of a combined "TRUNCATE SCHEMA [SCHEMA_NAME]" statement for all tables. Every statement must be processed by the database driver before execution in the database and every statement must create a string entity in the application server, send it to the database that executes the query, check and save the state of the query and return the result to the application server, which then re-checks the state of the query and continues with

the next one. While every query is fast in embedded mode, it still needs up to 1ms for execution. Due to the high number of tables, over 100 truncate statements are required, leading to 100ms maximum runtime to clean all the tables.

The third and best solution is to forbid any change to the tables that contain the master data sets. This requires refactoring of the test cases that perform changes to the master data sets. All tables containing master data sets were moved into a second schema and were referenced by synonyms with the same name from the standard schema. In this approach, all master data sets were isolated in a new schema, which can be access-restricted by controlling its privileges (i.e. READ-ONLY access). Thus, the master data sets cannot be manipulated by any test case and do not need to be re-initialized between tests. The tests that change the master data sets were refactored or were moved into their own test suite, which does not restrict access to master data set tables to READ-ONLY in the second schema, but instead executes a "TRUNCATE SCHEMA [SCHEMA_NAME]" statement on both schemas and re-initializes the second schema with the master data set. To reduce the test runtime, the test suite uses a controller that rotates already initialized databases on demand and performs re-initialization while tests are running on other databases.

Figure 1 shows the test system architecture, which ensures that every test has an initialized database by creating a new embedded database on demand. As initialization requires a lot of time, test runtime can increase dramatically. To counter this, multiple embedded databases are initialized on startup and a specified number of already-initialized databases is available from the test controller at all times. A database is automatically re-initialized in a background thread upon test case completion.

Figure 2 illustrates a sample configuration with three databases ($k = 0$ and $m = i_0 = i_n = 1$) where the index is rotating in accordance with the rule $i_m = 1 + \mathrm{mod}(m,3)$. When Test 1 has finished, Database 2 has been initialized, when Test 2 has finished, Database 3 is ready, etc. Note that the re-initialization of our testing system was 4 times longer than the average test case. Therefore, the number of idle databases should be high enough to counter this time gap. This may make rotational reprovisioning unappealing for test scenarios with short average test runtimes.

Switching to HSQLDB enables parallel test execution. The number of parallel runs depends on the number of CPU cores (processing units) on each workstation and the RAM (memory) consumption by a JUnit (the unit testing framework for the Java programming language) fork. The developer workstations have a CPU with 4 real cores and hyper threading and 32 GB of memory. When running tests in parallel, only a fraction of the tests is executed per JVM and the maximum memory requirement is reduced to about 4 GB for 3 or 4 parallel test suites. With the Java Garbage Collector, less than 4 GB is required for a fork. However, it is not recommended to force the JVM to use the minimal amount of required memory for its task. Using 4 parallel test forks with a maximum memory of 4 GB each requires up to 16 GB. In addition, the host operating system and installed services require about 8 GB of memory, and the development environment (like Eclipse with about 2.5 GB of memory), browser tabs and other applications also require more memory.

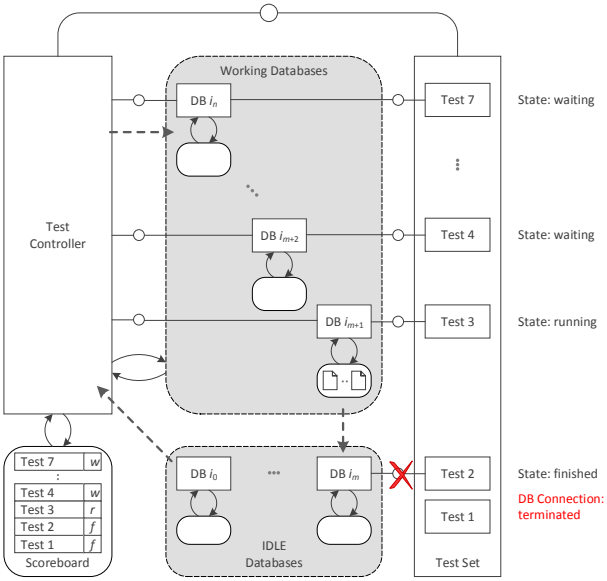

Figure 1. Test system architecture

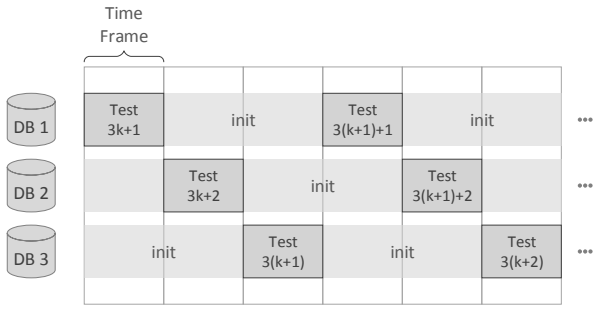

Figure 2. Rotational reprovisioning initialized databases

The main reason for using 3 or 4 parallel forks, however, is the amount of CPU cores available. Comparisons of runtimes with up to 8 parallel forks

indicate that increasing the number of forks above the number of CPU cores available is detrimental to the overall runtime. This is also highlighted by CPU utilization at test runtime: 60%-95% with 3 forks and 80%-100% with 4 forks (without much runtime performance improvement). Using multiple forks somewhat mitigates the previously described problem with increasing database connections, as every JVM requires its own database connection and this does not increase fourfold per fork. In our case, the final test runtime was reduced from 23 minutes in Oracle to 9 minutes in HSQLDB and to 5 minutes by using 3 or 4 forks.

6. Conclusions and future research

In this paper, we use the ADR methodology [26] to design, build, implement and evaluate a novel IT artifact – a database test architecture in a CI environment – for Germany's Federal Employment Agency. We describe the development of the architecture, the process used to migrate the existing four-year old large-scale application to the innovative test environment, and the design principles that shaped this migration and ensured the move could be done with a relatively small effort. The goal of the new in-memory test architecture was to improve the agency's CI practices by running each test for every commit made to the source code and ensuring that this quality gate step can be performed efficiently, without increasing testing time by an impracticable amount.

Based on our structured literature review, few studies address DevOps CI implementation in real-world database projects, and none provides a detailed analysis of the design choices, making this paper an important stepping stone for future studies in this area. This paper contributes to theory and practice by highlighting generalizable CI test architectures design principles for large database systems with complex code and test dependencies. Other organizations managing such systems can apply the process described in this paper to develop a new in-memory test architecture, select the most appropriate in-memory system, and parallelize the tests. The test environment design principles from section 4.2 (discrepancy between the test and production environments, CI requirements, and minimizing costs) and the corresponding in-memory database solution described in section 4.3 are readily applicable to other situations as well. The in-memory system selection will depend on an organization's specific circumstances - for example projects that are not as Oracle-specific as ours may prefer a different database. However, the selection criteria and the system comparison presented in section 5.1 can serve as a general template. Last, but not least, other organizations can adopt the design principles for implementation and the resulting test architecture presented in section 5.2. These procedures are very general and can apply to many projects that fall into the same category as ours – big, complex, developed over time (not just in the beginning development stages when architectural decisions are still flexible and open), and even mission-critical (where ad-hoc changes cannot be made easily).

This research has several limitations. We study a single implementation – although we discuss the technical features of the project in detail, we are not able to show how different organizational and technical context may shape the design. Very few academic studies published to date were useful in guiding our work – thus making the ADR process more dependent on the IT consulting and practitioner expertise of the co-authors of the paper. In addition, we were not able to document all stakeholder interests, values and assumptions – for example, we did not analyze the developers' attitudes towards DevOps CI.

Future research can explore the application of the above-mentioned general procedures in other organizations in different sectors and can identify similarities and differences across organizations. Future research can also analyze other database types such as NoSQL databases, distributed databases (such as Cassandra), time series databases (such as KairosDB or Prometheus), or graph databases (such as Neo4J). Future research questions might focus on why and how DevOps can be used with these "newer" databases, on the benefits of the approach, and on detailed use cases.

6. References

[1] Ambler, S. W., "Test-Driven Development of Relational Databases", IEEE Software, 24(3), 2007, pp. 37-43.

[2] Bärenfänger, R., B. Otto and H. Österle "Business Value of In-Memory Technology–Multiple-Case Study Insights", Industrial Management & Data Systems, 114(9), 2014, pp. 1396-1414.

[3] Baskerville, R., "Distinguishing Action Research from Participative Case Studies", Journal of Systems and Information Technology, 1(1), 1997, pp. 24-43.

[4] Baskerville, R., and M. D. Myers, "Special Issue on Action Research in Information Systems: Making IS Research Relevant to Practice: Foreword", MIS Quarterly, 2004, pp. 329-335.

[5] Birngruber, E., P. Forai, and A. Zauner, "Total Recall: Holistic Metrics for Broad Systems Performance and User Experience Visibility in a Data-intensive Computing Environment", Proceedings of the Second International Workshop on HPC User Support Tools, ACM, 2015.

[6] Chang, C.-R., M.-J. Hsieh, and J.-J. Wu, "HSQL: A Highly Scalable Cloud Database for Multi-user Query

Processing", Proceedings of International Conference on Cloud Computing, IEEE, 2012, pp. 943-944.
[7] DBMaestro, "2018 Database DevOps Report Examines the Release Management Process", 2017.
[8] Dodt, D., N. Cook, D. McDonald, D. Harting, and S. Pamela, "Improved Framework for the Maintenance of the JET Intershot Analysis Chain", Fusion Engineering and Design, 88, 2, 2013, pp. 79-84.
[9] Fernández-García, A.J., L. Iribarne, A. Corral, J. Criado, and J. Z. Wang, "A Flexible Data Acquisition System for Storing the Interactions on Mashup User Interfaces", Computer Standards & Interfaces, 59, 2016, pp. 10-34.
[10] Forsgren, N., J. Humble, G. Kim, A. Brown, and N. Kersten, "2017 State of DevOps report", Puppet + DORA, 2017.
[11] Gottesheim, W., "Challenges, Benefits and Best Practices of Performance Focused DevOps", Proceedings of the 4th International Workshop on Large-Scale Testing, ACM, 2015, pp. 3-3.
[12] Ghantous, G. B., and A. Gill, "DevOps: Concepts, Practices, Tools, Benefits and Challenges", Proceedings of Pacific Asia Conference on Information Systems, AIS, 2017.
[13] Hahn, G. J., and J. Packowski, "A Perspective on Applications of In-Memory Analytics in Supply Chain Management", Decision Support Systems, 76, 2015, pp. 45-52.
[14] Jabbari, R., N. bin Ali, K. Petersen, and B. Tanveer, "What is DevOps?: A Systematic Mapping Study on Definitions and Practices", Proceedings of the Scientific Workshop Proceedings of XP2016, ACM, 2016.
[15] Janes, A., and G. Succi, Lean Software Development in Action, Springer-Verlag, Berlin Heidelberg, 2004.
[16] Keijzer-Broers, W., L. Florez-Atehortua, M. de Reuver, "Prototyping a Health and Wellbeing Platform: an Action Design Research Approach", Proceedings of Hawaii International Conference on System Sciences, IEEE, 2016, pp. 3462-3471.
[17] Kitchenham, B., "Guidelines for Performing Systematic Literature Reviews in Software Engineering", EBSE Technical Report EBSE-2007-01, 2007.
[18] Lacoste, F. J., "Killing the Gatekeeper: Introducing a Continuous Integration System", Proceedings of the Agile Conference, IEEE, 2009, pp. 387-392.
[19] Lwakatare, L. E., T. Karvonen, T. Sauvola, P. Kuvaja, H. Holmström Olsson, J. Bosch, and M. Oivo, "Towards DevOps in the Embedded Systems Domain: Why Is It So Hard?", Proceedings of Hawaii International Conference on System Sciences, IEEE, 2016, pp. 5437-5446.
[20] Meyer, R., V. Banova, A. Danciu, D. Prutscher, and H. Krcmar, "Assessing the Suitability of In-Memory Databases in an Enterprise Context", Proceedings of International Conference on Enterprise Systems, IEEE, 2015, pp. 78-89.
[21] Miao, D., W. Chen, D. Tang, Y. Liu, and L. Jia, "Research on HSQLDB Concurrency", Proceedings of International Conference on Computer Application and System Modeling, IEEE, 2010, pp. V11-477-V11-481.
[22] Nori, A., "Mobile and Embedded Databases", Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM, 2007, pp.1175-1177.
[23] Ou, R., "Test-Driven Database Development: A Practical Guide", In Maurer F., D. Wells (Eds.), "Extreme Programming and Agile Methods - XP/Agile Universe 2003", Lecture Notes in Computer Science, 2753, Springer, Berlin, Germany, 2003.
[24] Plattner, H., and A. Zeier, "In-Memory Data Management: An Inflection Point for Enterprise Applications", Springer, Heidelberg, Germany, 2011.
[25] Rehmann, K. T., C. Seo, D. Hwang, B. T. Truong, A. Boehm, and D. H. Lee, "Performance Monitoring in SAP HANA's Continuous Integration Process", ACM SIGMETRICS Performance Evaluation Review, 43(4), 2016, pp. 43-52.
[26] Sein, M. K., O., Henfridsson, S. Purao, M. Rossi, R. Lindgren, "Action Design Research", MIS Quarterly, 35(1), 2011, pp. 37-56.
[27] Seltzer, M. I., and M. A. Olson, "Challenges in Embedded Database System Administration", Proceedings of the Workshop on Embedded Systems, USENIX, 1999.
[28] Sharp, J., and J. Babb, "Is Information Systems Late to the Party? The Current State of DevOps Research in the Association for Information Systems eLibrary", Proceedings of the Americas Conference on Information Systems, AIS, 2018.
[29] Soni, M., "End To End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, And Continuous Delivery", Proceedings of the International Conference on Cloud Computing in Emerging Markets, IEEE, pp. 85-89.
[30] Stolberg, S., "Enabling Agile Testing Through Continuous Integration", Proceedings of the Agile Conference, IEEE, 2009, pp. 369-374.
[31] Vaishnavi, V., W. Kuechler, and S. Petter (Eds.), "Design Science Research in Information Systems", DESRIST.org, 2004 (created in 2004 and updated until 2015 by Vaishnavi, V. and W. Kuechler; last updated by Vaishnavi, V. and S. Petter on December 20, 2017)
[32] van Deursen, A., "Software Engineering without Borders", Proceedings of the International Conference on Automated Software Engineering, IEEE, 2017, pp. 3-3.
[33] Virmani, M., "Understanding DevOps & Bridging the Gap from Continuous Integration to Continuous Delivery", Proceedings of International Conference on the Innovative Computing Technology, IEEE, 2015, pp. 78-82.
[34] vom Brocke, J., S. Debortoli, O. Müller, and N. Reuter, "How In-memory Technology Can Create Business Value: Insights from the Hilti Case", Communications of AIS, 2014, pp. 34-37.
[35] Zhang, H., G. Chen, B. C. Ooi, K. L. Tan, and M. Zhang, "In-memory Big Data Management and Processing: A Survey", IEEE Transactions on Knowledge and Data Engineering, 27(7), 2015, pp. 1920-1948.