

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/3249260>

Introducing Version Control to Database-Centric Applications in a Small Enterprise

Article in IEEE Software · February 2007

DOI: 10.1109/MS.2007.17 · Source: IEEE Xplore

CITATIONS

9

READS

176

4 authors, including:



[Wilhelm Hasselbring](#)

Christian-Albrechts-Universität zu Kiel

340 PUBLICATIONS 3,903 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Titan - Industrial DevOps [View project](#)



MooBench [View project](#)

Introducing Version Control to Database-Centric Applications in a Small Enterprise

Jan Ploski, *OFFIS Institute for Information Technology*

Wilhelm Hasselbring, *University of Oldenburg*

Jochen Rehwinkel and Stefan Schwier, *Meteocontrol GmbH*

When introducing new technologies, small software development enterprises must pay attention to version control.

When you own a small business specializing in data processing and Internet-based service delivery, your primary focus is building and publishing useful, visually appealing, and fast-responding Web applications. Open source technologies are both mature and widespread enough to let you achieve this goal without major financial investments in software licenses or processes.

Initially, creating a software engineering infrastructure and adhering to defined procedures seem less important than quickly delivering innovative services to customers. Therefore, your company's development activities center on rapid prototyping and are driven by immediate business needs. After you've established your brand and market share, your focus shifts to nonfunctional aspects of software development: the services must not only provide the added value your customers demand, they must also become trustworthy¹ and dependable.² You also wish to exploit emerging technologies to achieve your goals.

Slowly but surely, you realize that your needs have outgrown the simple development environment that originally served you well. You discover that your company's software quality assurance depends mainly on the uncoordinated efforts of individual developers. Dur-

ing an interaction with an external project partner, you realize that you've ignored version-control tools, and yes, you've suffered from typical configuration-management problems. Time is ripe for improvements. However, installing a central source-code repository and making sure everyone becomes familiar with a new tool isn't enough. If the software developed by your organization, like most current Web applications, stores its data in a relational database, the required versioning of the database schema along with the source code will likely pose a significant challenge. This article offers practical advice on methodically addressing the database version-control problems with simple tools.

We examine the software process-improvement challenges related to version control³ encountered by our project partner, Meteocontrol (www.meteocontrol.de), the German market

leader in Internet-based monitoring of solar power plants. Since October 2005, Meteocontrol has been involved in our research project WISENT (<http://wisent.d-grid.de>), which focuses on introducing Grid technologies and enhancing collaboration in the energy meteorology community. During this project, Meteocontrol must transform its centralized production and development environment to achieve better support for offline debugging and testing. The project-specific goal is to enable experimentation with new Grid technologies without disturbing existing business-critical services. However, database version-control issues are general enough to be of interest to any small company concerned with Web development.

The development environment

Meteocontrol is a small enterprise with a team of six software developers. The company aggregates performance data from distributed sensors and maintains several Web portals through which private and corporate customers can supervise their own solar panels. The company cooperates with the University of Oldenburg and the German Aerospace Center to process irradiance data based on satellite imagery.

Meteocontrol's online platform for content delivery consists of several networked servers located in a remote server park, as illustrated in figure 1:

- The Linux-based application server processes daily observation data obtained from distributed sensors installed in customers' photovoltaic modules and provides a PHP Web front end to the performance-monitoring engine.
- The Linux-based database server hosts multiple MySQL databases with a total of 240 Gbytes of measurement data, growing at a steady pace. It also maintains a smaller amount of data (2.5 Gbytes) on module configurations.
- The Linux-based geographic information server provides 2D map content and geolocation services to the application server.
- The Linux-based Web server holds Meteocontrol's static Web site and processes data imports.
- The Windows-based authentication server runs a proprietary user-management application.

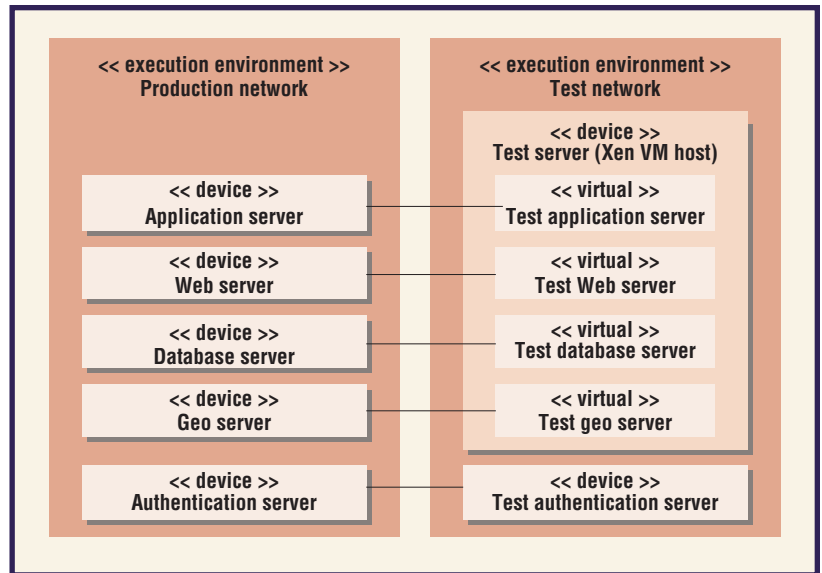


Figure 1. Overview of Meteocontrol's remote server park.

To reduce the risk of untested features becoming visible to customers through the production site, PHP code development occurs in a secondary "sandbox" location. Developers copy updated and new files from the sandbox to the main application manually once they consider them ready.

Observed deficiencies

Meteocontrol developers have been aware of the described mode of operation's disadvantages for some time. For example,

- developers can inadvertently overwrite each other's changes,
- changes have limited traceability,
- only authors can revert changes,
- deprecated or backup versions of files created during updates are accrued but forgotten after their successful incorporation into the production code, and
- safely performing major changes (such as software upgrades) is difficult without disrupting service operation.

Surprisingly, this list doesn't include the actual primary drivers for the decision to introduce sophisticated version-control procedures. Foremost, the increasing load on the application and database servers made a business case for process improvements. Specific to Meteocontrol's application, long-running data processing jobs must be executed overnight. However, because of the company's growing international customer base, the nightly maintenance

**Reverting
database
changes is
inherently more
complicated
than switching
to an earlier
source code
revision.**

window has been shrinking steadily. Finally, the marketing department's demand for data warehouse features (such as ad hoc reporting) has been increasing. The ability to completely disconnect the resource-demanding internal data analysis from the online transaction processing represents an additional competitive advantage for Meteocontrol.

Requirements for an improved development environment

Meteocontrol's primary requirement was the establishment of a test and development system that closely mirrors the production system's structure. To meet this requirement, the company purchased an additional multiprocessor server. Meteocontrol decided to map the production environment to virtual machines installed on the test server, emulating the actual hardware and software configurations. Virtualizing the machines in the test and development environment is a first step toward virtualizing the entire production system.

Further discussions with the developers revealed that the new server could also serve as

- a development platform,
- a testing platform for functional tests required before activating new features,
- a debugging platform for resolving issues in the production system, or
- a compute server for long-running queries.

Regardless of the combination of roles chosen for the new machine, it had to maintain the current process's agility in terms of rapidly incorporating new features and fixes into the production system. Meteocontrol deemed it inappropriate to introduce system-level version numbers and significantly longer release cycles because the system needed to flexibly respond to customers' short-term requirements.

Challenges and tradeoffs

It quickly became apparent that some of the requirements conflict. For example, for testing, debugging, and internal long-running data analysis, the production database should be continuously replicated to the new server. However, using the machine as a development platform requires allowing both content and structural alterations in the database, which can conflict with the replication.

Importantly, developers can't perform all de-

velopment tasks using a smaller local copy of the database with only a subset of data. Efficiently reproducing and troubleshooting problems encountered in the production system depends on access to the current, complete version of the database, as does processing long-running queries required by the marketing department. Finally, developers need a representative amount of data during development and testing to anticipate potential performance problems.

The propagation of the modified source code across the test and production systems and among developers is a well-understood process supported by today's version-control tools. Alas, the same isn't true for incrementally updating the database, especially when structural changes are concerned. Additionally, reverting database changes is inherently more complicated than switching to an earlier source code revision.

Introducing the test and development platform is thus fraught with risk:

- It increases the potential for inconsistencies.
- It increases the administrative overhead for committing changes.
- Developers must become familiar with new tools and techniques.

We realized that the above risks, if not dealt with carefully, could increase the number of defects in the production system.

Solution

To overcome the risks, we devised a tool-supported process to integrate source code and database schema version control with data replication. We deliberately kept the process simple and reliant on freely available technology. This let us

- serve developers who have no prior experience with formal software development processes,
- reduce the required amount of training and the negative effects on productivity during initial adoption,
- enable a smooth, step-wise transition from the current system architecture and working style, and
- increase the overall acceptance.

Our process of introducing version control consists of six steps, as figure 2 shows. At the

time of writing, we're nearing completion of step 3 in the Meteocontrol system.

Step 1: Duplicate production systems on virtual machines

We partitioned the new machine into Xen virtual machines⁴ whose file systems we created by copying disk contents of the respective production servers using Rsync.⁵ We needed a version of Xen with support for the Linux kernel 2.4, which imposed a 4-Gbyte limitation on the total system memory and indirectly on the number of managed virtual machines. We virtualized the Linux servers and reproduced the Windows server in hardware.

Step 2: Import source code into a repository

We installed a Subversion repository on one of the virtual machines to support version control.⁶ We transferred the complete source code and accompanying resources from the virtual machine representing the application server into the repository without any modifications.

Step 3: Improve configurability and clean up dependencies

The Web application is brought into a working state on the virtual machine. This step allows for (database-neutral) refactorings under source code version control. These changes aim to centralize configuration settings and remove undesired dependencies on hard-coded absolute paths and URLs. Making the application runnable in its testbed also includes enabling external access from Meteocontrol's offices. The virtualized servers connect to an internal network in which they use the same host names as the real servers; at the same time, when communicating with external clients, the application must properly translate this information, either directly by the PHP scripts or by a dedicated HTTP proxy.

Step 4: Use manual database version control

Once a working, independent replica of the production system becomes available on the test server, we adopt Subversion for transferring source code and database changes from the virtual machine to the real production server and vice versa. At this stage, we use the repository trunk to represent the production system's state, committing no untested changes. In other words, we recreate the sandbox environment on the virtualized server.

Ruby on Rails Migrations

Modern Web development frameworks provide basic support for database schema evolution. Ruby on Rails (www.rubyonrails.org), with its successful migrations feature, might currently be the most influential of these frameworks.

A *migration* is simply a Ruby script with two methods—up and down—which contain the operations required to upgrade the database to the next version or downgrade to the previous version. You can conveniently express most such operations in Ruby. Ruby also supports (but discourages) the use of native SQL. The scripts are incrementally numbered and committed to the repository along with the source code. Ruby on Rails maintains an integer version number in the database and executes one or more migration scripts to update the schema per the developer's request. This procedure is both simple and flexible.

However, it also has several disadvantages: it offers no way to use standard tools for database manipulation, and no efficient method to track the history of schema elements; developers can make mistakes when manually entering scripts; and script testing and conflict resolution in case of concurrent upgrades (for example, when using branches) is cumbersome.

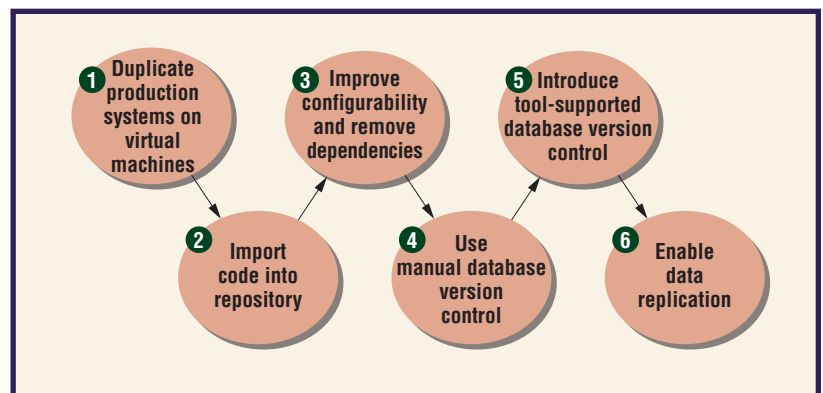


Figure 2. Step-wise introduction of version control.

Initially, we manually record database changes performed by developers as SQL scripts and check them in to the repository together with the source code. The production server executes these scripts after checkout. This approach resembles the schema evolution support that some frameworks offer (see the “Ruby on Rails Migrations” sidebar). However, it's still error-prone because it requires much discipline from developers. It's not easy to justify the additional effort compared to Meteocontrol's former working style. For example, testing an SQL script before committing it to the repository is cumbersome—the test database already contains the script's intended effects.

Furthermore, some of the PHP scripts can create data in the test database that should also be transferred to the production system. Developers must copy such data manually into

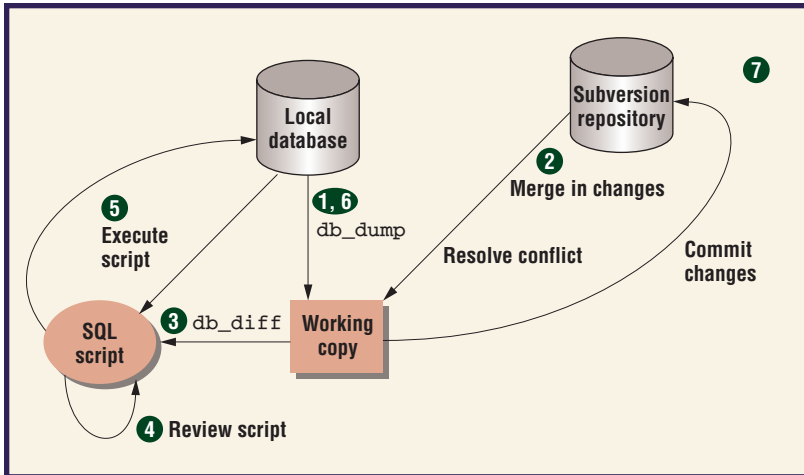


Figure 3. Data flow during update and commit operations with database version control.

the production database or recreate it on the production system by interacting with the PHP scripts there.

Step 5: Introduce tool-supported database version control

To remove the inefficiencies caused by manually editing SQL scripts, we develop two command-line tools for automating database schema version control: `db_dump` and `db_diff`. The general idea is to convert database schema elements into first-class configuration items managed by a simple version-control tool with only text-based diff capability.

The `db_dump` tool exports the database schema with the contents of selected configuration tables into a human-readable text format. It exports each data schema element's definition into a separate text file to support detailed tracking of changes. Thus, answering questions such as "who last changed table X?" becomes easy, unlike with the previous approach of representing changes as SQL scripts. The algorithm for transforming the database schema into text format must remain deterministic to prevent spurious change notifications.

Finally, to avoid false conflicts, `db_dump` must keep intact time stamps for files whose counterparts in the database haven't changed. (Subversion, like many other version-control tools, doesn't examine file contents to determine conflicts.)

The `db_diff` tool compares two sets of configuration items (dumps) created by `db_dump` and generates the SQL script necessary to transform the database from the state represented by one dump to the state represented by the other dump.

Figure 3 shows how these tools enhance the version-control process:

- Before every update from (1) and commit to (6) repository, developers invoke `db_dump`, so that their local working copies reflect their own, as yet uncommitted, changes in the database.
- During the update, Subversion automatically merges the database configuration items that were modified remotely into the local working copy (2). Developers resolve detected conflicts manually using standard text-based diff and merge tools, as they would source code conflicts.
- After the update, developers execute `db_diff` (3) to generate an SQL script that transfers changes into the local database (5). They can review the script (4) before its execution.
- Conflicts can also be detected during an attempt to commit changes (7). The committer will address these conflicts through an update cycle as described in step 2. Again, this closely mirrors the procedure for resolving conflicts in the source code.

Step 6: Enable data replication to support testing

The final step introduces data replication (using MySQL⁷) from the production server to the development server to support debugging and long-running queries. The replication will likely remain disabled for some tables to avoid collisions on independent updates of the production and test server (you can't entirely avoid updates on the test system while debugging and testing).

The actual development, which requires schema alterations, will stay on the same machine, but it will use another copy of the database, which must stay synchronized with the production system only as far as the schema is concerned.

Risks and limitations

The key success factor has been the proposed schema's acceptance by Meteocontrol's developers and management. In designing our approach to database version control, we tried to balance simplicity and low operational overhead with the generality needed to support various conceivable data manipulation tasks. Un-

der these constraints, we must deal with several limitations.

First, there's no support for arbitrary data migration. We can't automate arbitrary data-migration tasks (including many refactorings; see the "Agile Database Techniques" sidebar) using our planned tools. For example, to concatenate columns of an existing, nonempty table, a developer must perform a series of operations, including schema extension, data copying, and destructive schema modification. On the basis of the developers' feedback regarding the likelihood of such changes, we decided that developers must perform them manually. We can combine this process with the SQL script-based approach to version control to log changes.

In addition, implementing the tools can create substantial overhead. Implementing a correct SQL diff tool for MySQL isn't trivial because the order of applying changes to the database is restricted in nonobvious ways (for example, moving a primary key from one `auto_increment` column to another is a multistep procedure). Instead of building the envisioned tools from scratch, we can adapt an existing tool to our requirements. However, even this requires substantial testing. Because of the tool's role in updating a production database, we included the additional review step that the developer performs on the generated SQL script.

A third limitation is that our version-control schema assumes that both table definitions and selected tables' contents are exported as text and kept under version control. For performance reasons, this is only feasible for relatively small tables. Luckily, this restriction isn't important in our context.

Finally, the number of database copies restricts development. Using a single 240-Gbyte database limits our ability to implement multiple features concurrently. Even though we can cleanly separate the feature-relevant source code using Subversion branches, the same is not true for database schema configuration items, which we derive from only one database instance. Consequently, it would be difficult to update the production database when one feature becomes ready to ship without simultaneously importing changes belonging to other, unfinished features.

Making a complete copy of the 240-Gbyte database for each feature branch doesn't seem

Agile Database Techniques

With the advent of agile development approaches¹ and their strong emphasis on evolutionary software construction, database schema evolution has gained additional practical importance. Scott Ambler's book² and Web site (www.agiledata.org) contain an in-depth technical discussion that also covers various related topics, such as object-relational mapping. Moreover, they offer insight into challenges originating from the traditional division of roles among developers and data modelers.

There's also a growing interest in database refactoring, which itself is an important subtopic of database schema evolution. Ambler and Pramod Sadalage's book³ is recommended not just for tech-savvy developers, but also for tool vendors who should realize the database refactoring technology's market potential—not at all limited to small enterprises.

References

1. Agile Alliance, "What is Agile Software Development?" June 2006; www.agilealliance.org/intro.
2. S. Ambler, *Agile Database Techniques: Effective Strategies for the Agile Software Developer*, John Wiley & Sons, 2003.
3. S. Ambler and P.J. Sadalage, *Refactoring Databases: Evolutionary Database Design*, Addison-Wesley, 2006.

feasible due to hardware constraints. The likely solution will be to move all development to small, feature-specific databases. Developers will perform only the required performance tests and final functional tests on the actual test database, which will be locked for use by a single feature at a time.

Freely available software configuration-management tools aren't enough. Best practices for software configuration management in database-centric Web applications aren't freely available. This fact represents a major challenge for small enterprises—such as our partner—that simply can't adopt the practices of more experienced teams available in a larger company.

Joint research projects between academia and industry are beneficial for both parties and recommendable for small and medium-sized enterprises. Such projects offer companies an opportunity to gather feedback on their internal development process and start realizing longer-term goals with investments that would be difficult to accommodate in a day-to-day operative budget. Considering the role of small and medium-sized enterprises in our economy, this additional support is commendable. Importantly, the company management is typically responsible for establishing

About the Authors



Jan Ploski is a research staff member in the Business Information Management department of the Offis Institute for Information Technology in Oldenburg, Germany, and a member of the TrustSoft Graduate School on Trustworthy Software Systems. His research interests include software development processes, software quality assurance, and automated troubleshooting. He received his graduate degree in information systems from the Cologne University of Applied Sciences, Germany. Contact him at the Offis Inst. for Information Technology, Escherweg 2, D-26121 Oldenburg, Germany; Jan.Ploski@offis.de.

Wilhelm Hasselbring is a professor of software engineering, chair of the TrustSoft Graduate School on Trustworthy Software Systems, and a scientific director in the Offis Institute for Information Technology. His research interests include software engineering and distributed systems, particularly software architecture design and evaluation. He received his PhD in computer science from the University of Dortmund, Germany. He's a member of the ACM, the IEEE Computer Society, and the German Association for Computer Science. Contact him at the Univ. of Oldenburg, D-26111 Oldenburg, Germany; hasselbring@informatik.uni-oldenburg.de; <http://se.informatik.uni-oldenburg.de/Members/willi>.



Jochen Rehwinkel is head of the IT Department at Meteocontrol. His research interests include extreme programming. He received his degree in computer science and economics from the Augsburg University of Applied Sciences, Germany. Contact him at Meteocontrol GmbH, Energy & Weather Service, Spicherer Str. 48, D-86157 Augsburg, Germany; j.rehwinkel@meteocontrol.de.

Stefan Schwier is a software developer at Meteocontrol. His research interests include configuration management and software design. He received his degree in computer science and engineering from the Augsburg University of Applied Sciences, Germany. Contact him at Meteocontrol GmbH, Energy & Weather Service, Spicherer Str. 48, D-86157 Augsburg, Germany; s.schwierz@meteocontrol.de.




the required contacts with academia. These contacts naturally start as cooperations based on a common business goal, but they can evolve to include more general technology and process-related knowledge transfer.

Small organizations seldom assign a strategic value and allocate up-front investments to software engineering, especially if their business models don't consist exclusively of constructing software. Instead, individual developers advocate and introduce best practices when they're confronted with growing business requirements.

Open source software plays a major role in a small company's software development activities. Because of developers' grass-roots efforts in introducing new technologies, small companies are more likely to adopt improved practices and tools from the open source community than through the involvement of commercial consultants or vendors.

Moving away from a highly informal process that depends on individual developers

to make numerous case-by-case decisions and toward a more systematic, general approach isn't easy. Organizations must set appropriate expectations and carefully consider tradeoffs because short-term overheads are much more perceptible than long-term improvements. Finally, an organization must realize that any process automation effort must accommodate exceptions: it can't cover all process-related decisions, nor does it substitute for individual competence. Nevertheless, both developers and managers in small organizations should take interest in software engineering practices for automating mundane tasks and preventing trivial mistakes. Without this necessary step, they won't be prepared to face more serious, business-related challenges. 

Acknowledgments

German Federal Ministry of Education and Research (BMBF) sponsored our work on this publication under grant 01C5968.

References

1. W. Hasselbring and R. Reussner, "Toward Trustworthy Software Systems," *Computer*, vol. 39, no. 4, 2006, pp. 91–92.
2. A. Avizienis et al., "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Trans. Dependable and Secure Computing*, vol. 1, no. 1, 2004, pp. 11–33.
3. J. Estublier et al., "Impact of Software Engineering Research on the Practice of Software Configuration Management," *ACM Trans. Software Eng. Methodology*, vol. 14, no. 4, 2005, pp. 383–430.
4. Univ. of Cambridge Computer Laboratory, Systems Research Group, "The Xen Virtual Machine Monitor"; www.cl.cam.ac.uk/research/srg/netos/xen.
5. A. Tridgell et al., "Rsync—Faster, Flexible Replacement for Rcp," June 2006; <http://samba.anu.edu.au/rsync/>.
6. C.M. Pilato, B. Collins-Sussman, and B.W. Fitzpatrick, "Version Control with Subversion," O'Reilly Media, 2004.
7. MySQL AB, "MySQL 5.0 Reference Manual," chap. 6, 2006; <http://dev.mysql.com/doc/refman/5.0/en/replication.html>.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.