Crossroads News
A Monthly Publication for
Software and CM Professionals

Applying Agile SCM to Databases

Steve Berczuk, Steve Konieczka, Brad Appleton – January 2004

Many applications have database components, and these components evolve in the same environment as your code and other development artifacts. This article explores some of the issues around applying version management to database development in an agile environment. The article raises more questions than it answers, and we hope that it starts a dialogue about this important, yet often neglected topic.

Intro

We are often asked how to apply version management to databases. Developers are often at a loss when it comes to this problem for reasons that are technological and cultural. Also databases present a variety of problems related to version management, and one is often tempted to solve the more visible problems while overlooking simpler problems whose solution can provide much benefit. We'll discuss this in more detail below, but the next couple of paragraphs introduce the ideas.

Many of the technological issues are somewhat obvious. Source code is often managed at the module or file level; database changes are thought about at the logical entity level. There is a vast assortment of tools that can help us to detect changes and versions for text files; these tools don't seem to provide much benefit in the database realm. While there are tools that claim to help you manage versions of databases, it seems that they don't solve the problem for many due to cost, function, or ease of use, leaving many teams with no solution. It is important to avoid the temptation to ignore the problem because you don't have "the right tool." Your team will still waste time (and hence, money) working around the issues caused by the lack of any database version management process. You may be able to address your more critical problems by using approaches similar to what you are using to version manage source code.

The cultural differences revolve around the fact that databases are often "managed" by a group of people who are different that the development group, or who have a different set of skills. While logical database modeling is similar to, say, object modeling techniques, there are differences. Also, "database people" often have a "production mindset" that is different than developers; while organizations are getting accustomed to allocating resources for developers to manage their source code in private workspaces, applying that model to databases often meets resistance.

Often two domains are more similar that they appear at first glance. Since most of the people reading this article are familiar with version management principles for source code, there are many techniques that we use every day that can help us with databases. Since databases are closely related to the source code for an application, there are many benefits to starting with a unified approach and then looking for the differences. In this article we will talk about how to apply some of the same principles and patterns that you use for source code management to this other, slightly more complicated, problem. We will also provide you with some pointers to more information about agile database modeling.

This is a domain where we don't claim to have all of the answers. What we hope to do is to provide a basis for discussion, whether among yourselves, in the CM Crossroads Forums, or on the SCM Patterns list. Through this process we hope that we can guide you to an approach that is both simple and effective.

What's Different Technology and Culture?

Changes to databases are different than changes to code in a number of ways:

- Databases come with state; code changes only affect state when persistence is involved. If we change the database schema, we will consider tracking:
    - What changed in the schema, what columns were added, deleted, renamed. etc.
    - How to migrate the schema for an existing database from the old version to the new.

    - How to migrate the data from the old database to the new.

- Databases versioning is more complicated to manage. For code, we have many tools that allow us to take snapshots, store diffs, etc. Databases are more complex
- Databases are managed separately from the code. This does not make sense in many cases, since databases and code are closely related; the code has components that access the database, and the database structure is, in part, determined by what the code does. Working in a database application involves stepping across a chasm between the “Developer view” and the “DBA view” quite frequently.
- Database Administrators (DBAs) have traditionally worked from the mindset of a Systems Administrator. This mindset gets them thinking that they are simply "managing" a growing and changing system versus being involved in creating a reproducible "release" of an application. Many DBAs never create "releases" of their databases, they simply make changes to an existing database. This contrasts with the view one takes when managing source code where we build and release entirely new versions of components rather than alter existing ones. Changes to the database are also associated with the process of migrating existing data to the new schema. This is an important task, but you can view it as one that depends upon tracking changes to the schema rather than the actual act of tracking changes to the schema.

One approach to managing database changes is to have strict processes in place that control who can make changes, and when and how these people can make them. This is not an acceptable solution in an agile environment, since the database and code are often linked closely together, and bottlenecks in changing the database schema make your team less than agile.

What does it all mean?

We need to define what we mean by the phrase “managing versions of databases.” This phrase can mean at least two things, and often both at the same time:

- Managing how the “schema” (the logical structure of the databases) changes over time. Answering this question means that you need to understand some of the following issues:
    - When did a table/column/etc get added or removed?
    - What in the schema changed between last week and today and who made the change?
    - How do I change the structure of an existing database to match the current one?
- Managing how to migrate the data in an existing database into the new schema. This is, or course, essential in a production environment, but is arguably something that you can do with the help of a change management system.

A related task is managing changes to the database data at various points in time, but this is the domain of backup and recovery practice, rather than “version management” practice, so we will not address it here.

The most basic function you need is the ability to track and maintain changes to the schema, and to reproduce an earlier version of the database schema, in particular, one which goes with an earlier version of the application code. A straightforward way to do this is to manage the artifacts of the database.

Versioning the Artifacts

One approach to tracking changes to the schema is to track changes to some artifacts that we use to create and manage the database. This seems analogous to versioning the source code of an application, rather than the executable components.

The key artifacts for a database contain Data Definition Language (DDL) statements, which specify how to create the tables/columns etc. From a DDL file you should be able to create a new, empty, database, from nothing. A file containing Data Manipulation Language (DML) commands specifies how to insert important data into the tables when you have data that must exist before the application accesses the database.

With the correct version of these artifacts, and with the appropriate scripting in place, any developer should be able to recreate the version of the database that goes with a version of the codebase. Also, these artifacts serve as a starting point for answering questions about what changed between releases.

At first glance, versioning only artifacts such as DDL files seems too simplistic to solve any real problems. Yet it is an essential first step. It is also an important step in moving your database in line with your agile code development process.

Being Agile

At first glance, databases are very different entities than source code, so you may feel that your standard toolbox of tools and patterns for managing changes to source code may not be up to the task of managing versions of your database components. Consider, however, that your goals for managing your database are very similar to your goals for managing your codebase. Also, consider that your database and your code depend on each other, so there are many benefits to using a similar approach. If there are too many differences, and the database changes happen with a different pulse, your database will be a bottleneck. To be agile, you need to help things work smoothly.

In an agile environment, developers can always recreate the latest version of the application from what is in the version management system, and they can test changes to their code before committing it to the codeline. These things should also be true of databases.

Others have described approaches to using databases in an agile environment. Scott Ambler's site: http://www.agiledata.org/ has a number of useful articles and links about data modeling in an agile environment. Scott's book, Agile Database Techniques is a useful reference. The article *Evolutionary Database Design* by Martin Fowler and Pramod Sadalage (http://www.martinfowler.com/articles/evodb.html) provides a good overview of how to work with databases in an agile environment. This article will focus on the version management tasks.

The patterns from the Agile SCM pattern language are relevant to the database domain as well as to the code domain. The implementation has additional complications, since a database is more involved than source code, and we don't have the tool support that we are used to for source code.

Consider how the following patterns apply to Agile Databases:

- **Active Development Line**: The database is part of the application, and changes to the database should not be a bottleneck. You need to be able to make changes to a schema, and make the changes available to other developers.
- **Private Workspace**: being able to experiment with a change before releasing it to the rest of the team is an essential mechanism for encouraging change, while also providing for stability. You need to be able to create your own database that you can alter, test with, etc without interfering with the work of others. To do this, you should move towards having each member of your team having their own database that they can experiment with changes to the schema, and recreate the tables at will.

- **Private System Build**: you need to build and test your application with your version of the database. Being able to create a version of the application, including any database components.
- **Smoke Test, Unit Test, Regression Test**: you need to test both the application with the new schema, and any issues relating to the new schema.
- **Release Line**: There may well be a production schema and a development schema. If you have a release line for your code, your database should evolve in a similar manner.

Adopting this approach requires a change in mindset from databases as "controlled by the DBA" with changes made to the database, to the database as an active part of the development process.

Conclusion

Databases present their own special issues. Code can often be thought of as flat files. Databases can't.

Rather than doing nothing because you don't have the best solution, consider adapting basic SCM practices. We'd like to hear what your thoughts are about this issue. Email us, or discuss this in the General CM forum on CM Crossroads.

Brad Appleton is co-author of **Software Configuration Management Patterns: Effective Teamwork, Practical Integration**. He has been a software developer since 1987 and has extensive experience using, developing, and supporting SCM environments for teams of all shapes and sizes. In addition to SCM, Brad is well versed in agile development, and cofounded the Chicago Agile Development and Chicago Patterns Groups. He holds an M.S. in Software Engineering and a B.S. in Computer Science and Mathematics. You can reach Brad by email at *brad@bradapp.net*

Steve Berczuk has been developing object-oriented software applications since 1989, often as part of geographically distributed teams. In addition to developing software he helps teams use Software Configuration Management effectively in their development process. Steve is co-author of the book **Software Configuration Management Patterns: Effective Teamwork, Practical Integration**. He has an M.S. in Operations Research from Stanford University and an S.B. in Electrical Engineering from MIT. You can contact him at *steve@berczuk.com*. His web site is www.berczuk.com

Steve Konieczka is President and Chief Operating Officer of SCM Labs, a leading Software Configuration Management solutions provider. An IT consultant for 14 years, Steve understands the challenges IT organizations face in change management. He has helped shape companies' methodologies for creating and implementing effective SCM solutions for local and national clients. Steve is a member of Young Entrepreneurs Organization and serves on the board of the Association for Configuration and Data Management (ACDM). He holds a Bachelor of Science in Computer Information Systems from Colorado State University. You can reach Steve at *steve@scmlabs.com*