Bringing Database Development into your Agile Processes

Using automation to shorten the database development cycle

by John Pocknell, Senior Manager - Product Management, Dell Software

Introduction

Once an organisation becomes convinced of the advantages of agile development, there is no going back. Shorter time to value, lower risk and greater flexibility are among the most frequently cited benefits of the agile mentality.

Although most application developers now consider agile a mainstream approach, database developers—especially those working on relational databases—have been slower to embrace it because of the need to understand and respect the state of a database when deploying changes. Thus, database professionals have had to rely on manual processes that do not scale up to the faster development cycles at the heart of agile.

This presentation explores what it will take to bring database developers into the agile fold. It describes agile as it applies to database development and emphasises three points:

1. People don't scale in a linear fashion, so the manual processes used now in database development will never scale to the level required to support agile projects.

2. Mostly, those processes are slow, manual and methodical because of the high risk involved in making a mistake on a production database.

3. The way to both reduce risk and scale up is by automating as much of the testing, review and staging processes as possible.

The main takeaway for application developers, database developers and database administrators (DBAs) is that automation is the key to synchronising the development cycles of application software teams and database teams.

Agile development for reduced risk

Agile is a reaction to old-school, rigid, "waterfall" methods of building software. Such traditional methods rely on deep specification, design, documentation and months or years of heads-down work to deliver a product. They seek to reduce risk through careful and methodical attempts to deliver a complete and perfect product, which takes a long time. But because business requirements and customer landscape often change while all that work is underway, the company ends up launching its product into a very different market and missing the mark. Frustration ensues.

Agile principles emerged in an attempt to correct this, proposing instead:

- Close collaboration between the programmer team and business experts
- Face-to-face communication (as more efficient than written documentation)
- Frequent delivery of new, deployable business value
- Tight, self-organising teams
- Ways to craft the code and the team such that the inevitable requirements churn was not a crisis[1]

Without rigidly enforcing specific practices, agile embodies the best of these principles as a mindset for delivering better software faster. Development teams can deliver value and improvements in smaller chunks more frequently, instead of saving all the value until the end of the project. When software development processes are properly automated, the result is actually less risky, even when teams are putting out releases more often, as depicted in Figure 1.

[1] Kent Beck et al., "The Agile Manifesto," *Agile Alliance,* 2001, *www.agilealliance.org,*

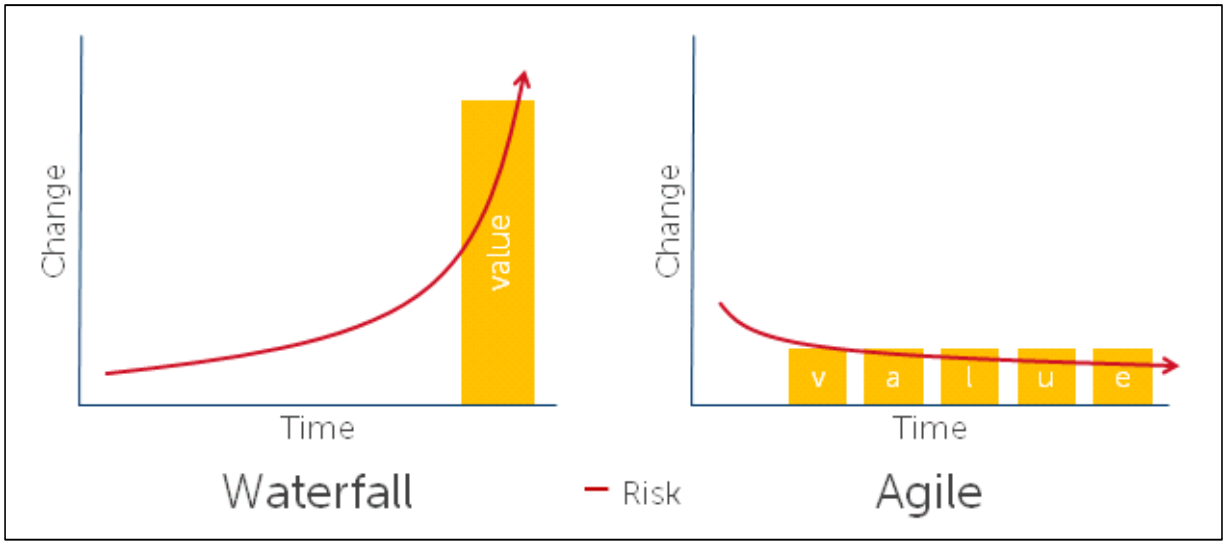

Figure 1: Shorter time to value, lower risk

Since its inception in 2001, agile has caught on among application developers. In 2009, Forrester found that 35 percent of the 1,298 IT professionals surveyed were using agile development methods.[2] By 2014, agile tool vendor VersionOne found that 94 percent of 4,000 companies surveyed practiced agile development, and 53 percent characterised the *majority* of their agile projects as successful.[3]

[2] Dave West and Tom Grant, Ph.D., "Agile Development: Mainstream Adoption Has Changed Agility," *Forrester*, January 20, 2010, http://programmedevelopment.com/public/uploads/files/forrester_agile_development_mainstream_adoption_has_changed_agility.pdf.

[3] Press release: "VersionOne Releases 9th Annual State of Agile Survey Results," *VersionOne, Inc.*, March 26, 2015, http://www.versionone.com/about-us/press-releases/article/VersionOne-Releases-9th-Annual-State-of-Agile-Survey-Results/

What about agile database development?

One of the principles that agile organisations follow is to "deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale." Application software developers have successfully adopted that principle, resulting in shorter development cycles.

But what about database developers? Why have they been slower to adopt agile practices and shorter development cycles? The answer lies in a few important differences between application development and database (especially relational database) development:

Overwriting

Application developers write code for execution, and they can easily overwrite version 1 of that code with version 2. If there's a problem with version 2, it's easy enough to restore version 1 so that people can continue using the application in its previous state while the developers fix version 2.

But a database is more like a living organism with a current state that must always be managed to maintain the integrity of the data. It's not an option to overwrite version 1 of the database schema with version 2 because that could result in data loss. Instead, database changes must be made through scripting, telling the relational database management system (RDBMS) to adjust its state from the current form to the intended form. Thus, it's harder to post a change to a database, and if the change doesn't work or data is lost, then it becomes necessary to restore the entire database, which results in costly system downtime.

Version control

Few software development teams can do without version control because it prevents multiple developers from working on the same source code at the same time and allows them to reconcile any differences. It represents a single version of the truth.

Database development teams, however, use it differently; if they use it. The database itself is regarded as the master version of the source, and its state must be maintained throughout the deployment process, so version control does not play the same role, in spite of the frequency and extent to which changes are made to stored procedures and functions. This fundamental difference between application and database development increases overall risk and makes it more difficult to implement automated deployment mechanisms.

Automation

Application developers use tools to automate the entire deployment process—version control, builds, unit testing, static code analysis, staging and even deployment—so they can overwrite previous versions of code and deliver new code to production multiple times each week or day. And their extensive use of automation for things like building and testing causes their level of risk to decrease, as shown in Figure 1.

To protect the data contained in the database, database developers and DBAs have to manage its state through scripting that gracefully updates the database from one state to another. However, the scripting process is often devoid of automation, slowing the overall release process.

Urgent changes

That lack of automation at the database level exacerbates matters when urgent fixes need to go into production. Application teams can easily get the fixes into the pipeline for the next release, with agile processes ensuring that they will undergo proper testing, unit testing and staging before go-live. But the database team must manually walk changes through a much longer cycle.

Without agile, changes of any kind go through the database development cycle much more slowly. That's not a problem, as long as the business can wait. But what if the business finds a critical bug in a procedure or function that requires an immediate change? In many cases, the production DBA will have to make the change and apply it directly to the live production environment with minimal, usually rushed testing (see Figure 2).

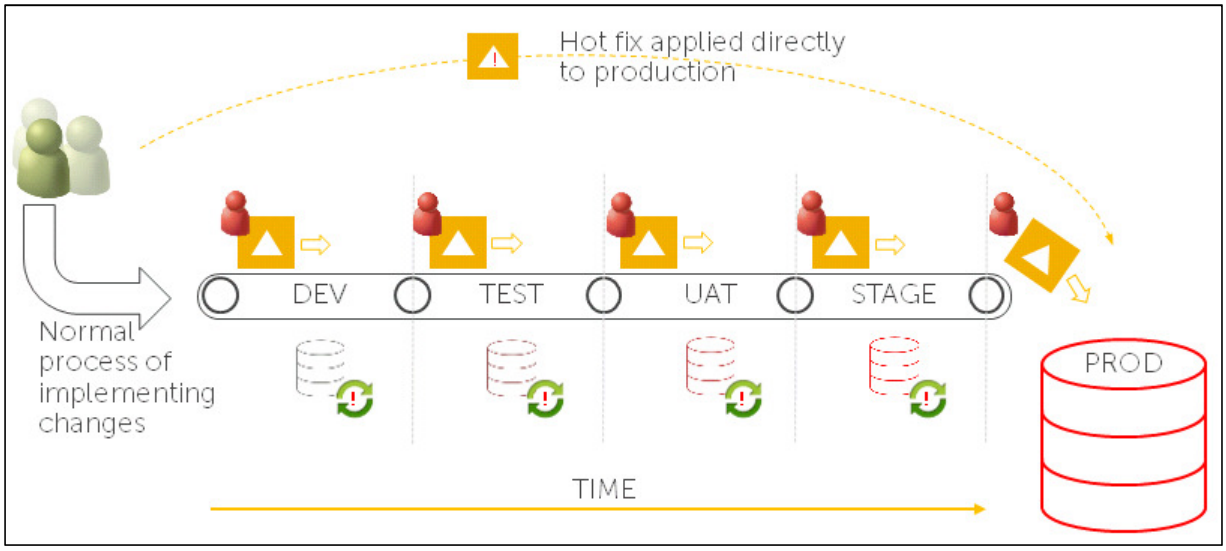

Figure 2: Urgent change required by the business

While that may solve the immediate concern, it knocks the other upstream environments—development, QA, unit testing, stage—out of sync with production. That increases risk for the work underway on them and requires yet more manual work to bring all the environments back into harmony with one another. As disruptive as this is, it's necessary to resolve the business issue.

There must be a better way.

Scaling to shorten the development cycle

Those manual processes do not scale in a linear fashion. After a certain point, adding more people will actually slow development down and lengthen the development cycle.

Consider a team of five people working on database code changes. Their average development cycle is three weeks. To shorten the cycle down to two weeks they add two or three developers, for a total of seven or eight. Next, they're under pressure to shorten the cycle to a single week. If they add three people, will they accomplish that? Possibly, but 10 or 11 developers working on the same code base makes for a lot of cooks in the kitchen, especially at such a breakneck pace.

Now suppose they need to shorten the cycle to one day to catch up with the application development team. Adding five to ten more people—even if they could find and afford them—would almost certainly result in chaos and significantly more management overhead than progress. It would be almost impossible for the team to deploy daily in the sensitive database environment. When teams reach the point of diminishing marginal returns, productivity begins to decrease as people are added. This is not a number-of-people problem.

Still, in the world of application development, a single-day cycle is not extraordinary. In fact, many application teams deploy directly to production hundreds or thousands of times each day because they've automated those pieces of testing and integration that often take so long. Database development teams will never scale to the speed required to keep up with agile application development teams unless they introduce software automation into their processes as well. In fact, until database development processes—particularly development on relational databases like Oracle, SQL Server and DB2—scale up to a similar level, they will continue to be the bottleneck in an otherwise agile organisation (Figure 3).

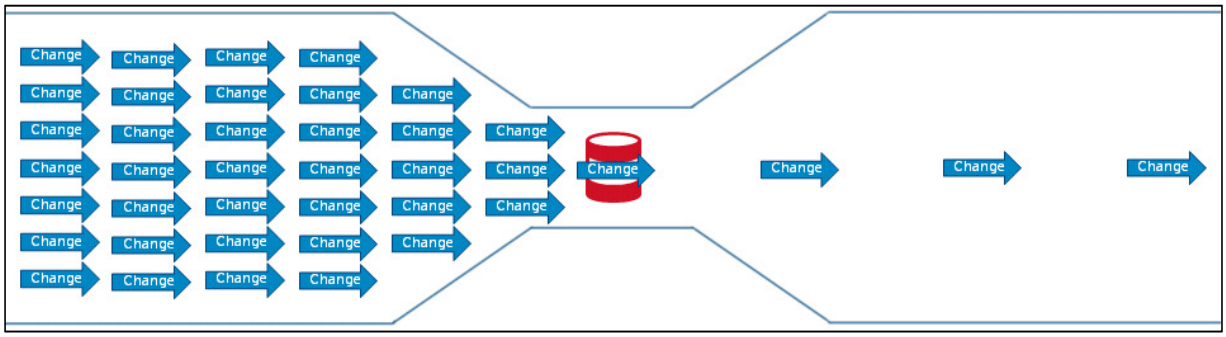

Figure 3: Relational database as the bottleneck in the agile organisation

Scaling database development for agile

Relieving that bottleneck requires an automated pipeline with which database development teams can address risk, ensure quality and shorten the development cycle for the organisation as a whole. Figure 4 depicts the essential pieces in that pipeline:

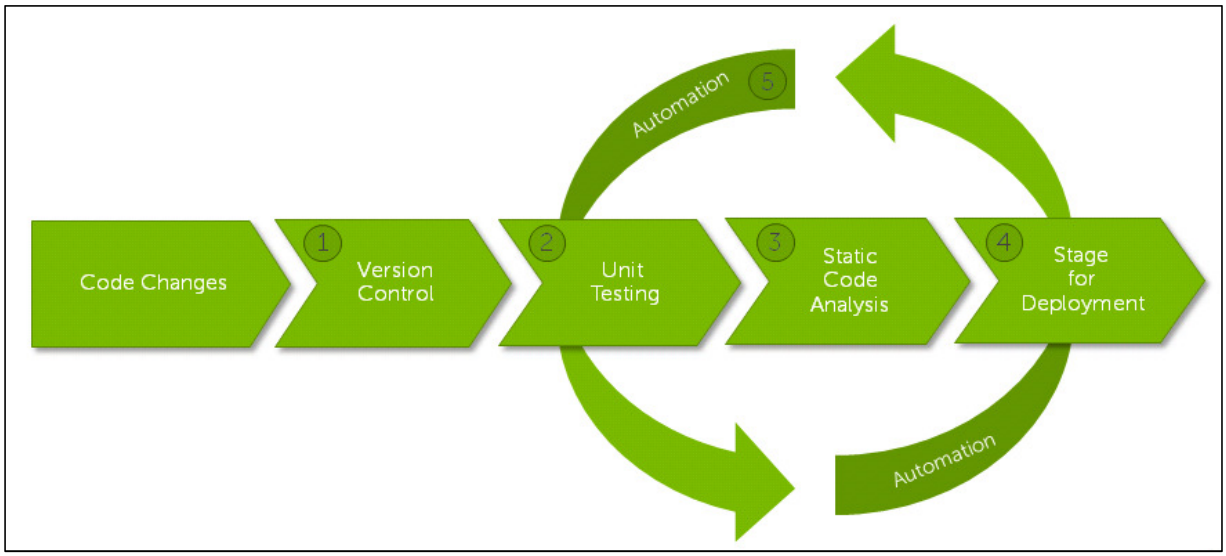

Figure 4: Redefining database development

1. Version control

As noted above, database development teams often forgo version control, but it is a valuable way of tracking data definition language (DDL) changes every step of the way. Tracking all code changes and revisions over time makes it easier to assess them, compare them side-by-side and diagnose any problems that arise on the path toward production.

2. Automated unit testing

The only way to be certain that a proposed change does not break something else is through automated unit or regression testing. Even if developers and QA teams are comprehensive and remember everything they need to test, manual testing is still vastly slower than automated tests that run against the code change at the point of check-in.

Automated testing provides a basis and a safety net for accelerating database deployments because it offers nearly immediate assurance that new changes do not break earlier work. Furthermore, by automating and executing this process at the point of check-in, developers will receive immediate feedback about a break and be in the best position to fix it quickly, when the relative cost to fix a bug is at its lowest (Figure 5).

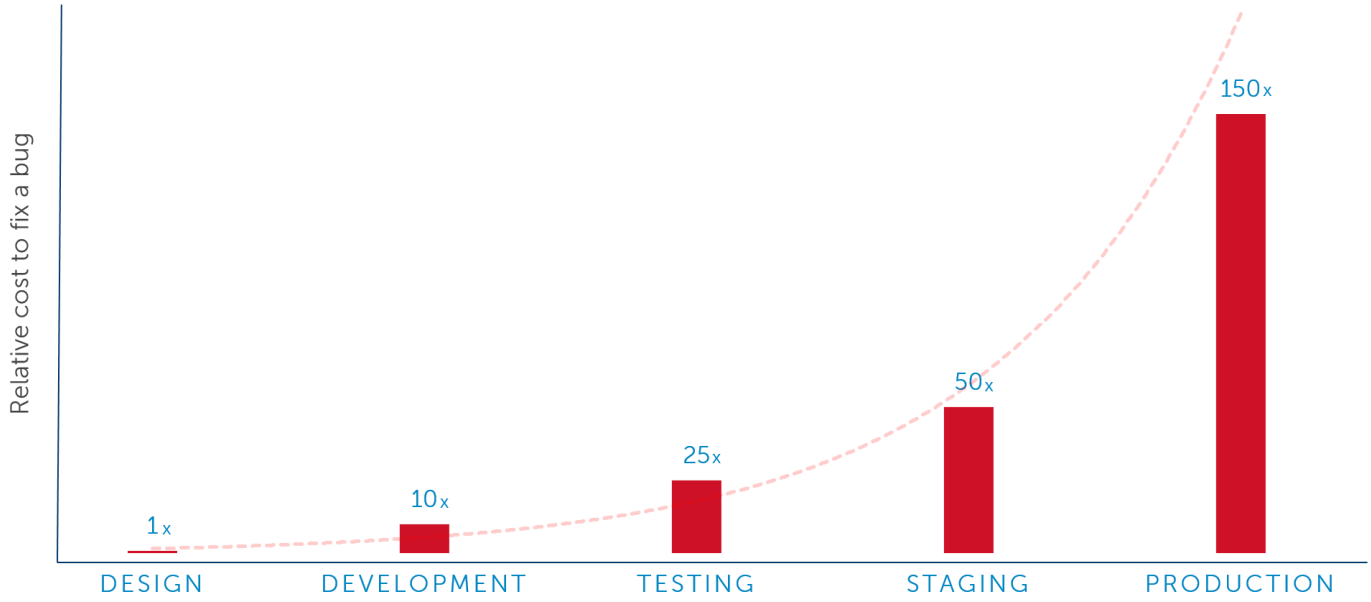

Figure 5: Cost of Fixing Bugs at Various Stages of Software Delivery

3. Static code analysis

Developers subject their code to peer review to ensure that they haven't missed something, introduced a security vulnerability, made a mistake in logic or inadvertently slowed the product down. Static code analysis software significantly accelerates this process and ensures adherence to company standards by reading the code and identifying the same kinds of patterns that peer developers look for.

Database developers, too, stand to gain from applying static code analysis after unit testing, perhaps even more than their application development peers. Given the extreme pressure and ever-tightening deadlines of the database development cycle, many teams rush the code review process or skip it entirely. Thus, automating the process at the database development level can reduce the time the team spends and increase overall consistency of code reviews. Automated code review can check against rules written to help enforce company standards and improve quality, performance, maintainability, security and flow.

4. Stage for deployment

To reduce the risk of data loss and other mishaps, the database development path usually includes a DBA stop in the deployment stage, for a review of code changes before they go into production. While automating around the DBA stop is not a good idea, automating the creation of ALTER scripts for deployment is a valuable step in shortening the development cycle.

DBAs managing the path to deployment can use automated tools to collect all the queued changes that have passed regression tests and static code analysis, compare them to the production environment and generate the scripts to commit them. Not only can this improve

DBA efficiency and shorten the development cycle, but it can also ensure all of a project's changes make it into production.

5. Automation

As the organisation moves away from manual processes and toward automated tools, database development cycles will start shrinking and all teams can begin to realise the promise of agile. Using software tools in a piecemeal fashion each step of the way is faster than pure manual processes, but it provides only incremental improvements.

Agile requires a fundamental shift in the way teams work. Instead of having people manage each step of the process, teams must look for ways to create fully automated pipelines that submit changes to regression testing, review and staging for deployment without further interaction from the team. DBAs can rest assured that code changes meet quality standards and adhere to company policy, and managers can see that code will meet project requirements and run properly in production.

Conclusion

Application developers have long taken advantage of agile practices to shorten their development cycle and reduce the risks associated with change. Database development, on the other hand, has traditionally relied on manual processes that reduce the risk of data loss in a live production database. The resulting bottleneck, especially in a relational database environment, has kept the organisation as a whole from realising the full promise of agile: the ability to release software in prompt response to market changes.

Given that human effort doesn't scale in a linear fashion, the way to relieve that bottleneck is to automate as much of the software development lifecycle as possible, particularly time-intensive and repetitive processes such as testing, review and staging of changes. Excellent tools exist for application developers; database developers, if sufficiently inspired by the agile mentality, can adopt them for use in their projects as well.

Toad for Oracle can integrate with modern continuous integration and delivery software to automate once-manual tasks, paving the way toward truly agile database development and deployment.

Learn more at software.dell.com/products/toad-for-oracle

About the author

John Pocknell

Senior Manager - Product Management

Information Management Group

Dell Software

John Pocknell is a senior manager of product management at Dell Software. Based at the European headquarters in the U.K., John is responsible for the strategy and roadmap for the Toad portfolio of products worldwide. He has been with Dell Software since 2000, working in the database design, development and deployment product areas, and he has run many Toad training courses for customers. John has spent the last 15 years successfully evangelizing Toad to customers at various events throughout Europe, the U.S. and APJ, and he writes many blogs and papers on the Toad user community, Toad World as well as Technical Briefs about Toad on the Dell Software website.

John has worked in IT for over 30 years, most of that time in Oracle application design and development. He is a qualified aeronautical engineer with more than 10 years of experience in provisioning IT consultancy services and implementing quality assurance systems to ISO 9001.

Contact Address:

18 Moat Road

Loughborough

Leics LE11 3PN

England

Telephone: +44 (0)1509 215325

Email: john.pocknell@software.dell.com

For more information

About Dell Software

Dell Software helps customers unlock greater potential through the power of technology — delivering scalable, affordable and simple-to-use solutions that simplify IT and mitigate risk. The Dell Software portfolio addresses five key areas of customer needs: data center and cloud management, information management, mobile workforce management, security and data protection. This software, when combined with Dell hardware and services, drives unmatched efficiency and productivity to accelerate business results. www.dellsoftware.com/.

If you have any questions regarding your potential use of this material, contact:

Dell Software 5 Polaris Way Aliso Viejo, CA 92656 www.dellsoftware.com

Refer to our Web site for regional and international office information.