WHITE PAPER

Datical

# The Future of DevOps:

## *Aligning Database Change Management with the Software Development Process*

# Introduction

There's a new emphasis on delivering smaller releases more frequently without sacrificing quality. Delivering in this way is less wasteful, allows the business to be more responsive to the immediate needs of its customers and allows for easier course correction caused by shifts in the market. As a result, organizations that rely on proprietary database applications to achieve their goals are undergoing a dramatic transformation in how they design, build and deliver those applications.

But the revolution that's currently happening in Enterprise IT is nothing new. In the late 1940s, Toyota's Taiichi Ohno and Eiji Toyoda set out to design a less wasteful manufacturing method that was pulled along by interactions with all suppliers, producers and consumers involved in the process. Over the next 30 years, Ohno and Toyoda established and continually refined what came to be known as the Toyota Production System.

The system's underlying principles, known as the Toyota Way, will sound familiar to anyone with even a passing familiarity with movements like Agile or DevOps.

- **Continuous Improvement** - The Toyota Way employs a long-term vision, meeting challenges directly and creatively addressing them to realize that vision. Business operations are constantly evaluated and improved with innovation and evolution as primary goals. To ensure maximum effect, decisions are made based on facts retrieved directly from the source: the consumers and producers of the business product.
- **Respect for People** – Participants in the process strive to build mutual trust and understanding. Participants take responsibility for their role because they understand the entire process and the importance of their contribution to it. An emphasis on working together to maximize individual contribution and team performance leads to opportunities for growth, both personal and professional.

The success of the Toyota Production System and the value of the Toyota Way is hard deny. Businesses of every size have found that embracing continuous improvement and respect for people yields efficient, consistent and adaptive business operations that can separate them from competitors or launch them into an entirely new weight class.

# Evolution of the Software Development Lifecycle

At its essence, producing software isn't radically different from manufacturing any other consumer good. It starts with a market need and goes through a period of design. The design is reviewed and the cost of production is determined. Finally, the product is built

and released to market. Feedback from consumers and the ever-evolving needs of the organization uncover new products and services that mark the beginning of the next cycle of production.

Way back when the Waterfall software development methodology ruled the earth, these cycles of production were each measured in months. All told, the gap between identification of the market need and delivery of the solution could be more than a year.

This was sufficient when the consumers' expectations were aligned with the slow rate of change. Today is a different story. Technology has spread into every corner of our lives. We have incredible access to the producers of the goods and services we rely on. The biggest barriers to acquiring new products and services are the distance between the consumer and their device and the strength of their Wi-Fi or data network signal. It is very much a, "What have you done for me lately?" economy. The demand for new and better features is enormous and the stakes are high. Any delay your organization has between the request for a service and its delivery could be all the delay a customer needs to swap you out for a competitor.

To keep pace with the evolution of technology and the impact it's had on the consumers' expectations, we've seen the rise of movements like Agile Development and DevOps. When you read the Agile Manifesto or any literature on the Three Ways of DevOps (The Phoenix Project is a notable example) you don't have to look very hard to see the DNA of the Toyota Way. Agile and DevOps are more of a shift in mindset than a prescription of tools, practices and process. Both emphasize communication and collaboration over strictly documented and siloed processes. They are also designed to collect feedback from all stakeholders frequently and to incorporate that feedback quickly to the benefit of all. Identification of waste in all its forms is paramount in order to continuously improve our methods and our offerings.

15 years after the release of the Agile Manifesto, we have well defined patterns for implementing Agile at scale. Descendant movements have sprung up like Continuous Integration, DevOps and Continuous Delivery. Each of these movements were accompanied by a huge leap forward in automation capabilities that support the application development lifecycle from end to end. In the most advanced shops a developer can now commit a change that triggers a fully automated delivery process. The code is automatically compiled and packaged; static and dynamic quality assessments occur throughout the process; test environments or new containers are generated on demand; the application is deployed and promoted through integration, test and production environments; all activities are thoroughly tracked and communicated to all stakeholders. In the case of a well-designed and acceptable change, there is virtually no human interaction or hands on time between the code commit and the push to production. Cycle times have decreased exponentially.

# Databases are Left Out in the Cold

The management of database change in support of these application changes is the notable outlier in this brave new world of automated construction, validation and deployment. Today, database change management is truly the last mile of the application release process and works more or less like it did 20 years ago. This has more to do with the protections we put in place to preserve and protect our data than anything else.
When it comes to tolerance for change and risk, databases are a completely different animal than an application binary or web server configuration. Databases cannot be simply reverted or replaced. The data inside them is mission critical to the organization and must be preserved. That's why production databases were walled off from development decades ago and highly skilled DBAs were installed as guards and gatekeepers for the company's most precious resource. They were instructed to resist change unless it could be definitively vouched for as safe and absolutely necessary.

Unfortunately, this has led to one of the most crucial application components being left out of the agile revolution. What's worse, the advancements in other areas have monumentally increased the velocity of change. The manual processes employed by the data team are not suited to fast paced agility. As a result, many data teams are buckling under the strain and the database has become the last bottleneck in the release process and the primary limiting factor in how fast an organization can actually move.

# The Future of DevOps and the Application Release Process: Embrace the Database

The disconnect between modern application development and the database change management process that supports it must be corrected for organizations to fully realize their investments in faster methods of delivery. We must rethink the data team's practices and methods and bring them into alignment with the proven practices established over the last 15 years by application development and release teams to safely speed delivery. This will not only increase overall release velocity but will also lead to a deeper understanding of the production process for all participants. The result is a distributed but cohesive team that understands the entire process, their role in it and their responsibility to one another. That understanding can then be the source of further evolution and growth in service to all.

At Datical, we build our solutions with the goal of fostering this deeper understanding and facilitating the alignment of these two processes that have been traditionally decoupled.

Our vision for our solutions revolves around what we refer to as the Five Pillars of Effective Database Automation. The Pillars represent what is required to truly bring database change management practices in line with modern application development and IT operations practices and bring the database into its rightful place as a tier one component in the application stack.

# The First Pillar: Package

More often than not changes in the application and changes in the database are made by two different people with two different contextual understandings. The application developer responsible for managing change in the application code is often provided with requirements and designs from the business. They typically have a good understanding of why the change or new feature has been requested and what the impact of that change is to the rest of the product. They translate the requirements and design into functioning pieces that satisfy the requirements of the business.

Database changes, on the other hand, are usually handled by DBAs who are given little to no explanation of why the change is being made or the larger impact of the change. They are simply informed that the change is necessary and either write or are provided with a script that affects the change. They are then asked at varying times to run that script in one or more environments over time until the change is ultimately pushed to production and fades into history.

Without a deeper understanding of the application or business needs that the script is associated with, they cannot effectively plan for releases in an agile manner. If a feature falls out of a release or a bug fix is pulled into an earlier release to address a customer issue the DBA lacks the contextual information necessary to quickly modify the plan. Instead they must piece together which scripts support the modified release on the fly and hope they deploy everything they should and don't deploy anything they shouldn't.

The first Pillar, **PACKAGE**, is intended to address this problem. The first step in packaging a database change is to tightly associate it with the development effort it supports. By tying changes to their features, DBAs are able to associate the activity with the driver more quickly and trace whether or not all of the appropriate schema components for an application change have been applied. The second step of packaging is then tying the features to the appropriate release vehicle. For example, if a particular application change is slated for a named 'JULY_20xx' release the supporting database changes for that feature should be labeled for this release as well. This allows the DBA to better understand and

participate in the application release process, especially as modifications to the established plan are made. If all changes are tightly associated with their feature and release, it becomes much easier to move those features in and out of releases without missing anything or grabbing too much.

## The Second Pillar: Validate

Application developers usually employ several layers of automated validation to assure that the changes they are making work as expected and don't break anything else inadvertently. There will be an initial layer of static code analysis that can identify bad coding practices or the possible introduction of performance constraints based on pattern recognition. Then there are automated unit and integration tests to verify that the code is working as expected with its dependent components. Finally there is some level of automated validation of the entire application to make sure that everything is as it should be and that the product is ready for consumption.

The primary limiting factors of current database change management in relation to release velocity are the expertise it requires to ensure changes are safely made and how time intensive it is to apply that expertise to every change that's proposed. Applying automation to address the time-consuming nature of this process has historically been viewed skeptically because of the complex knowledge required to accept or reject a proposed change. The only way to overcome this skepticism is to apply a method of automation that is designed to address the unique challenges posed by validating database changes.

## Jidoka: Automation with a Human Touch

The Toyota Production System includes a concept that perfectly describes what's needed: Jidoka or "automation with a human touch." The focus of this type of automation is not to replace a simple function, such as SQL script execution. It is geared more towards evaluating the process as it flows and stopping it when a condition that requires further review is detected. The Second Pillar, **VALIDATE**, is focused on implementing Jidoka to speed the process of change validation without sacrificing the safety inherent in manual reviewed by a skilled and trusted professional.

Recovering from a bad database change is difficult if not impossible. To avoid making a bad change in the first place, there are really two levels of automated validation that must occur. The first is confirming the structural soundness of the proposed changes. Over time, databases serving the same application can drift out of sync due to their persistent nature and the manual process of updating them. Perhaps a manual change was made during a troubleshooting session and not propagated to other environments or a script dropping a column from a table was accidentally skipped in the manual execution process earlier

in the development cycle. To compensate for these possible differences the first layer of validation should confirm that the database is in the appropriate state to successfully receive the changes. For instance, if one of the proposed changes is to add a column to a table the validation automation should make sure that the table exists and that a column of the same name does not exist. Once those two data points are affirmed the change should proceed without an issue.

But just because a change completes successfully doesn't mean it's a good change. The trickier part of validating a change is making sure it adheres to best practices, organizational standards and compliance requirements. To provide that in an automated fashion, a fully customizable validation mechanism must be present that harnesses the knowledge and expertise of the DBAs and applies that knowledge in an automated fashion to every proposed change that passes through the system. In our flagship product, Datical DB, we implemented a Rules Engine that allows DBAs to configure automation to find the same things they look for when reviewing a change. If a change does not violate any policies in the rule suite, the change passes through the system unhindered and the DBA saves precious time that can be applied to more strategic undertakings than reading SQL. However, if a change does violate a rule, the system is stopped and the DBA is alerted. After further investigation, the DBA can accept or reject the change and the process starts back up again. By only involving the DBA for manual review when it's absolutely necessary, change moves through the system faster without sacrificing quality or security.

## The Third Pillar: Deploy

Thanks to the advancements in Application Release Automation (ARA) in recent years, deployment of the application is a push button affair. With a single command or click in Jenkins, IBM uDeploy or CA Release Automation (formerly Nolio) the application environment is configured and the application deployed. Employing an ARA solution also has the added benefit of standardizing promotion processes. So the way an application is moved from Dev to Test matches the way its moved from Test to Stage and so on. By the time the release to Production arrives it's a non-event because the application and its deployment have been tested multiple times.

The Third Pillar, **DEPLOY**, is the concept that database deployment should be just as easy as the process described above. The current practice of manually processing a batch of scripts and triaging the errors that arise until everything is eventually successful must stop. The new deployment process should be simple and identical in every environment in the lifecycle to ensure the same validation and consistency that enables other application components to be released worry free. Within Datical DB this is partially accomplished by employing the first two pillars (Package and Validate) to ensure completeness and validity of the changes to be deployed. Once that's taken care of, it's one-click simple.

# The Fourth Pillar: Know

Having a complete understanding of what's happened in the application lifecycle is designed into every aspect the development process. Source code control solutions keep an accounting of modifications to the application code. Workflow management tools track the activity, progress and status of a release. The build system and automated tests include extensive logging and archiving. ARA solutions keep a similar accounting of the deployment process. When a release is complete there is a wealth of knowledge to support audit or process refinement tasks. Nothing is left to question.

Tracking of database changes is nowhere near so sophisticated. The execution of scripts is usually recorded manually in a spreadsheet or in a service ticket. Collating and validating this activity is extremely difficult. Proving that a database is at the appropriate version is typically an exercise in tracking down changes that can be tied to a specific release. Is there a "NICKNAME" column in the "USERS" table? Then it must be version 3.22!

As we've identified in other aspects of the process, tracking methods are largely manual. This introduces the possibility of human error and prevents us from truly trusting the various artifacts we use to piece the puzzle together. To bring the tracking of database change to the same level of sophistication enjoyed by the application development portion of the team, tracking must happen automatically. This is what the Fourth Pillar, **KNOW**, is all about. As we design our solutions, we look for every opportunity to collect and store data when a managed environment is materially changed. Removing the reliance on human action to track these changes leads to tracking data that is high quality, consistent and complete just like the data collected during application development, build and deployment.

# The Fifth Pillar: Integrate

Another key concept in the Toyota Production System is the concept of burden. A participant's burden is what they must do to fulfill their responsibility in the process. By ensuring that no participant is overburdened, we avoid the bottlenecks and mistakes inherent in a lopsided process. When a tool or practice adds significant complexity to the overall process the burden of participation in the process is increased for someone. In this case, the tool is overburdening one role to make gains in another.

The tool chains employed by application development groups are as varied as they are powerful. There are a wide variety of IDEs, source code control solutions, infrastructure providers, build systems and deployment offerings. No matter what individual products make up a group's chain of delivery the integration between them is seamless. The combined whole must be greater than the summation of its parts.

The Fifth Pillar, **INTEGRATE**, reflects that our tools should collaborate as we do and avoid overburdening others. When choosing a database change management solution for agile enablement you shouldn't be forced to limit your options for tools that handle other areas of the process. Information should flow freely between the tools and the tools should be easily invoked by any and all orchestration platforms so you have the flexibility to implement or change any other link in the chain.

## Conclusion

The road that will take us to the Future of DevOps and finally align the database with the rest of the software delivery process is a well-travelled one. Fittingly it was an automobile manufacturer that first paved that road. The lessons learned and principles employed there have since driven modernization efforts in other forms of manufacturing, including the development of software. The same things that have been accomplished there can be accomplished in the more difficult and delicate world of database change management. The key to success is learning from those that have gone before us, adopting their proven principles and committing to the specialized application of them. We must be open to feedback and collaboration. We must always seek to improve. Only then can we truly travel the last mile in release automation.

Datical

www.datical.com | info@datical.com | Tel: 949.DATICAL