CASE STUDY | PHARMACEUTICAL

How we learned to stop worrying and love continuous integration

with the SQL Automation Pack

redgate

ingeniously simple

" Our ROI has been high, not because we are writing more code, but because we are doing less maintenance, and encounter fewer conflicts in work. This enables us to have a larger team with more separate lines of development than might otherwise be possible."

ANNETTE ALLEN AND DAVE GREEN
Database developers, First Databank

Annette Allen and Dave Green are database developers at First Databank (FDB™). Annette has 12 years of SQL Server experience, and Dave has been working with SQL for the last 14 years. In this case study, Dave and Annette talk about their development environment and how to use Red Gate tools together with TeamCity continuous integration as an approach to running database projects.

A good time for a new solution

Before we started this project, we didn't use continuous integration at all, and we had two source control systems in place. AccuRev SCM was used by our software developers for internal maintenance tool development in C# code, and database projects were based in a customized system which had its roots in Visual Source Safe.

Our customized system had custom scripts and jobs to make it work with our database system which made it prone to error, difficult to use, and not very well integrated. It was basically unchanged since it was used with SQL 2000 and we wanted to be able to leverage the enhancements in more recent versions of SQL Server. We found ourselves with a good opportunity to step back and review our whole approach.

Our goals for the project

- Find a solution which allowed multiple developers to work seamlessly together on a sample piece of code.
- Be able to reuse our existing NUnit-based testing for continuous integration.
- A system that supported a planned move to embrace Test Driven Development, but not mandate it.
- A solution much more integrated into the design environment to minimize window swapping and the opportunity for forgetfulness.
- Use one source control system for database and application code.
- Give Management visibility so they can understand what we were doing on a day to day basis, how we were doing against known goals, and get a read on progress and status at a glance.

First steps

We needed to decide on one source control system going forward. Microsoft had made known its plans to discontinue support for Visual Source Safe so that option was out. AccuRev met our needs and we already had all the licenses we needed which meant there was no further cost, so we decided to move forward with AccuRev to source control our database and application code.

We also decided to try TeamCity continuous integration, a client-server based system with a web interface. TeamCity allows us to call existing NUnit testing frameworks, and there's a lot of information on the web to help you get started. We were also happy to be able to continue using Red Gate's SQL Source Control, SQL Compare, SQL Data Compare, SQL Test, and SQL Packager.

Which development model was right for us?

We'd previously worked with a shared development model and suffered with it. People were trying to change code at the same time which caused conflicts and we never knew which version was the most up-to-date. On the plus side, the Shared Development model has lower licensing costs, only one copy of test data and it's in a managed environment.

The alternative

With a dedicated development model you get your own sandbox environment. You're developing against something that is a known good so you don't have to worry about pre-existing issues. You can try it, break it, get a new copy and carry on, not affecting anyone else. Also, it integrates well with our AccuRev source control system.

We decided to go with a dedicated development model.

Our code promotion model

Our code is written in stages, we call them areas. When we're finished in one area, we promote the code to the next stage when other tests and decisions are made.

1. We start by developing locally, writing our code and running tests.
2. When we check in changes, the batch file gets promoted up to Shared Development where colleagues are able to work on it, including application developers, database developers, and testers. In the Shared Development area you'll find bits of code that have been started but not complete and some code that's complete.

   We promote the complete code that we want to put into a build from the Shared Development area into the Integration area.
3. In the Integration area we introduce code to the continuous integration server. The Integration area has automated testing for all the code we've put together and it's the first time we get to make use of automated integration testing. At the Integration level the formal set of tests is the primary indicator of whether the code is ready for the product.
4. The next area is acceptance, and that should only have working code that's made it through Integration. The code is put through an automated build and that's where the product managers decide whether it makes business sense to take the code to the customer.
5. When code gets to staging it's something that's ready to be released, a set of new features for the new version of the product.
6. In the final stage we build a package, we use SQL Packager to produce this, and then deploy it to live. Some people use release scripts at this point instead of SQL Packager, it depends on your infrastructure.

Using TeamCity with Red Gate tools

TeamCity allocates work by project. Within a project, you have builds for every stage of code - so there might be a staging build, acceptance build, and integration build. In the TeamCity UI you can see the tests associated with each build, whether they've passed or failed, how long they took, how long ago they ran, etc.

We can see the build steps associated with each configuration. Our builds tend to include a list of command line steps which we can dive into using Red Gate tools from the command line in SSMS.

One of the reasons we wanted continuous integration was so that we could have Test Driven Development and be able to do continuous unit testing. We run unit tests by calling a stored procedure. Using the tSQLt framework with SQL Test, we can use sqlcmd to connect to our database server and run the stored procedures. We can run the tests in SSMS and see whether the build passes from the TeamCity UI. To display the time for tSQLt tests to execute in TeamCity uses a small patch to the tSQLt framework – see the discussion.

In any build, our final step is a custom script that shows the test results to make it easy to see how the tests performed and which ones have failed. Without this step, TeamCity only shows you whether the build passes or fails, and it's not very user friendly to go through the build log and try to find out why it failed.

We also use muted tests within the TeamCity UI so that if a specific test fails, it doesn't cause the whole build to fail. This is useful if you have test data written before the code is written, or if you have a low-value feature with a known issue that hasn't been fixed yet.

Once our build has passed all its tests, we can decide whether we want to move it to the next stage in development.

Issues from a Development standpoint

As we rolled out continuous integration to more projects, we encountered a couple of issues. We were encouraging the development team to write a lot more tests than earlier on in the development process. As you build more tests, the runs take longer because there's more to do, so then testing might take 20-30 minutes, and if there are a lot of projects going at the same time, you can be sitting in a queue for a while. We found that using more build agents in TeamCity solved this.

We also transitioned to an agile environment which meant we were releasing more often than we had previously, which took some getting used to. We hadn't sufficiently planned for the amount of time it would take to get used to the new development process.

Security Models must be explored when working in teams with rapid deployment, we found it easier if we used the same permissions in continuous integration as in live. Ideally, you want to develop under similar permissions to live, and you certainly want the continuous integration system to work under live permissions, so that all permissions issues are encountered in testing.

We found we needed a Build Master, someone who is in control when promotions happen and when builds happen to make sure things go up atomically.

Results of moving to continuous integration

We keep the TeamCity UI visible on a large monitor; this lets management see at a glance how various builds are progressing. With this, we get better engagement with corporate sponsors.

The software development cycle is much shorter, though this is in part due to switching to agile development at the same time.

We got used to checking our code builds frequently. This makes for better code because we're checking smaller pieces and the code we're looking at is fresher.

More testing is done early on so bugs are found earlier, and we are also able to use our NUnit tests.

Development teams work together on smaller bits of code so we have a much better understanding of what everyone's doing and what people's problems are.

Our ROI has been high, not because we are writing more code, but because we are doing less maintenance, and encounter fewer conflicts in work. This enables us to have a larger team with more separate lines of development than might otherwise be possible.

About First Databank (FDB™)

FDB provides clinical drug knowledge that helps healthcare professionals make precise medication-related decisions. FDB is the only provider of clinical decision support to receive NHS Evidence accredited provider status from NICE.

FDB drug knowledge forms a critical part of primary and secondary care patient administration systems, medicines optimization solutions.

With thousands of customers worldwide, FDB enables our information system developer partners to deliver a wide range of valuable, useful, and differentiated solutions. We offer more than thirty years of experience in transforming drug knowledge into trusted, purposeful and effective solutions that improve patient safety and healthcare outcomes.

For further information please visit fdbhealth.co.uk or email us info@fdbhealth.com.