Agile Practices in Database Design Process

Prakash Badhe

prakash.badhe@vishwasoft.in

Non-Agile Database Design

- When deploying changes to database leading to the vulnerable state of the database has prompted the database designers to avoid implementing agile methodologies in the database.
- Their insistence on manually inputting changes has created a bottleneck in otherwise agile organizations, and this leads to a point where the database is becoming notorious for holding up application releases.

Database being Agile ?

- Most of agile practices revolve around dev and ops management.
- Database operations tend to be manual and slower.
- The results of database actions are usually tested from application front end UI services
- How to make Agile practices in data design and implementations..
- How to automate the database operations in DevOps context…

Test Driven Data ?

- DbUnit is a JUnit extension targeted at database-driven projects
- The DbUnit helps to put the database into a known state between test runs.
- DbUnit has the ability to export and import the database data to and from XML datasets
- With DbUnit integration tests are supported for different SQL databases.

Unit testing the Data

For SQL database and with some stored procedures, and a layer of code sitting between your application and the database, how to define tests in place to make sure that the code really is reading and writing the right data from the database?

The DbUnit includes a comprehensive built-in database test suite of test cases.

Steps for Data Testing

- Set up database
- Write code to access database
- Run code, do a SELECT and check if the records showed up in the database
- With manual or v**isual inspection** you don't do it often, and you don't check everything every time.
- Automated tests — painless tests that run often and test a lot — reduce the chances of the data is going missing

Test Data and Live Data

- The testing data in the database should not “mess up" live data.
- Multiple databases..unit test and production data.
- A good test that creates all the data it needs.
- Testing is simplified if you can get the database in a known state before a test is run.
- The test cases clean out the database before starting any tests.
- Define the tests to verify the data.
- Define re-usable queries and stored procedures to manipulate the generic data which can be parameterized.

Database refactoring

- A database refactoring is a simple change to a database schema that improves its design while retaining both its behavioral and informational semantics.
- The Database refactoring does not change the way data is interpreted or used and does not fix bugs or add new functionality.
- Every refactoring to a database leaves the system in a working state, thus not causing maintenance lags, provided the meaningful data exists in the production environment.

Code refactoring vs. database refactoring

- A database refactoring is more difficult than a code refactoring.
- The code refactoring only need to maintain behavioral semantics while database refactoring also must maintain informational semantics.

Why data refactoring..?

- To develop the schema in parallel with the design of the rest of your system.
- To fix design problems with an existing legacy database schema
- Database refactoring are often motivated by the desire for database normalization of an existing production database, typically to "clean up" the design of the database.
- To implement what would be a large (and potentially risk) change as a series of small, low-risk changes.

Database Refactoring Examples

- Splitting an aggregate table into two different tables in the process of code splitting.
- Renaming an existing column to make its purpose clearer.
- Combining two columns into a single one because they were being used for the same purpose.
- Splitting an existing column into two or more columns because the original column was being used for several purposes (so you have one column per purpose).

Refactoring Examples..

- In micro-service applications define separate database per service.
- Applying a common data format to a column so as to increase the consistency of the data.
- Common code refactoring (Rename Method, Introduce Variable, Rename Variable, and so on) to database code such as stored procedures and triggers.
- Introducing a view to implement a common access path to data.

Database Refactoring Process..

- The process of database refactoring is the act of applying database refactoring to evolve an existing database
- There are three considerations that need to be taken into account:
- How to implement a single refactoring
- How to track/share database refactoring across the organization
- How to apply a series of database refactoring to a database

Database refactoring tools

- LiquiBase : open-source database-independent library for tracking, managing and applying database schema changes
- SQL enlight : Detect Defects, FORMAT AND refactor the T-SQL Code.
- SQL Prompt : Library to Refactor SQL code.

Database getting agile..

- The database developers can implement the database agility safely and efficiently.
- This supports the database to scale up to faster development cycles demanded by the agile methodology.
- How to build an automated pipeline in which database development teams can address risk, ensure quality and shorten the development cycle for the organization as a whole .?

Agile Database

- **To execute agile database methodology** safely
- **Maintain Database Version Control:** The deployment script should be aware of database dependencies, ensuring the database code is covered.
- The deployment script being executed should be aware of the environment status when the script is executed.
- **Automated Testing :** Automated testing provides a safety net for agile methodologies for the database because it assures that new changes do not cause errors for earlier work.
- Automating the testing in early pipeline phases, when development and testing rapidly iterate, accelerates delivery and frees the resources to focus on creating value.

Techniques for agility

- **Static Code Analysis**
- Developers subject their code to peer review to ensure that they haven't introduced a security vulnerability, made a mistake in logic, or inadvertently slowed the product down.
- Static code analysis software 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.

Review..

- The review of code changes before they go into production.
- DBAs 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.
- This improve DBA efficiency and shorten the agile database development cycle, and ensure all of a project's changes make it into production.

Automation and Compliance

- 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.
- The 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.

Agile Data(AD) Methodology

- The Agile Data Method subscribes to the values and principles of the ***Agile Alliance***.
- The Agile Data (AD) method defines a collection of strategies that IT professionals can apply in a wide variety of situations to work together effectively on the data aspects of software systems.
- The AD as a collection of techniques and philosophies that enables IT professionals within the organization to work together effectively when it comes to the data aspects of software-based systems.
- Agile Data method goal is to describe a philosophical foundation on which an effective approach to the data-oriented activities on software development projects, and within IT departments, can be based.

Agile Data philosophies

- **Data:** Data is one of several important aspects of software-based systems.
- **Enterprise issues:** Development teams must consider, and then act appropriately, regarding enterprise issues. They need to take the bigger picture into account if they're to avoid building yet another silo application.
- **Enterprise groups:** Enterprise groups (such as enterprise architects, data management, ...) exist to nurture enterprise assets and to support other groups, such as development teams, within your organization. These enterprise groups should act in an agile manner that reflects the expectations of their customers and the ways in which their customers work.

Agile data...

- **Uniqueness**. Each development project is unique, as is each organization, requiring a flexible approach tailored to its needs, and therefore the relative importance of data varies based on the nature of the problem being addressed.
- **Teamwork**. IT professionals must work together effectively, actively striving to overcome the challenges that make it difficult to do so.

Sweet agile data..

- **Sweet spot**. You should actively strive to find the 'sweet spot' for any issue, avoiding the black and white extremes to find the gray that works best for the overall situation.

Master Data Management

- The Master Data Management (MDM) promotes a shared foundation of common data definitions within the organization, to reduce data inconsistency within the organization, and to improve overall return on the IT investment.
- MDM, when it is done effectively, is an important supporting activity for service oriented architecture (SOA) at the enterprise level, for enterprise architecture in general, for data warehouse(DW)/business intelligence (BI) efforts, and for software development.

Agile MDM

- Addresses basic MDM activities
- Is collaborative
- Is embedded within the development process
- Is an enterprise activity
- Is evolutionary
- Is usage-driven
- Produces measurable results
- Delivers quality through testing
- Adopts a lean governance approach
- Requires a cultural shift

Agile Data Best Practices

- These "best practices" for evolutionary/agile database development
- Database refactoring
- Agile data modeling: model the data aspects of a system iteratively and incrementally, with an agile approach in a highly collaborative manner and no more "big up front modeling (BMUF)".
- Database regression testing

Test first.. design

- To ensure that the database schema actually meets the requirements for it, and the best way to do that is via testing.
- With a test driven development (TDD) approach write a unit test before you write production database schema code, the end result being that you have a 100% regression test for your database schema.
- Agile testing provides the concrete feedback that you need to ensure data quality.

Configuration Management

- The configuration management of database artifacts.
- The data models, database tests, test data, and so on are important project artifacts which should be configuration managed just like any other artifact. The tools like Ansible, Chef, Puppet can be utilized for automation of the configuration.

Data for the Developer

- Developer sandbox: The developers need their own working environments, called sandboxes, where they can modify the portion of the system which they are building and get it working before they integrate their work with that of their teammates.

Data normalization

- Data normalization is a process in which data attributes within a data model are organized to increase the cohesion of entity types.
- In other words, the goal of data normalization is to reduce and even eliminate data redundancy, an important consideration for application developers because it is incredibly difficult to stores objects in a relational database that maintains the same information in several places.

Realistic Primary Key

- Set a realistic primary key strategy.
- Sometimes it makes sense to use natural keys and sometimes surrogate keys.
- Need to understand when to apply each strategy, and to be prepared to refactor if you discover that you've made the wrong choice.

Database encapsulation

- A database encapsulation layer hides the implementation details of your database(s), including their physical schemas, from the business code.
- In effect it provides the business objects with persistence services – the ability to read data from, write data to, and delete data from – data sources.
- Ideally the business objects should know nothing about how they are persisted.

Common development guidelines

- Having a common, usable set of development standards which are easy to understand and to comply to can greatly improve the quality of the systems that you develop.
- These guidelines may include, but not be limited to, programming guidelines, modeling style guidelines, data naming conventions, and user interface conventions (including report design conventions).

Lean data governance

- The goal of data governance is to ensure the quality, availability, integrity, security, and usability within an organization.
- The goal of agile/lean data governance is to enable development teams to do these things effectively within your overall IT ecology.
- The Lean governance is focused on enabling people and motivating them to do the right things.
- A lean data governance approach promotes a healthy, collaborative relationship between data professionals and the teams that they're supporting.

Thanks to you!!