Automation of QA in the project of DB migration from SQL Server into Oracle

Iakov Kirilenko & Eduard Baranov
Software Engineering Department,
Mathematics and Mechanics Faculty,
Saint-Petersburg State University,
Saint-Petersburg, Russia

Abstract— **This paper describes an automatic QA organization experience in the industrial project of DB migration from MS SQL Server 2005 to Oracle 11gR2. The resulting DB of the project is supposed to contain the same data and to have a functional correspondence with the initial one. The initial DB is quite huge: 6 terabytes of data and 2500 KSLOC of stored procedures. The documentation for the initial base is incomplete and outdated and doesn't correspond with the database in question. Functional specifications for stored procedures are missing, as well as tests. This article contains the description of the main problems solved during the project, solutions and an estimation of their applicability based on implementation experience.**

Keywords - database, data migration, testing, reengineering, quality assurance

I. INTRODUCTION

The authors are contracted to migrate the industrial database from MSSQL Server 2005 to Oracle 11gR2. There are main requirements to the project: all data must be migrated from the initial base and all functionality must be preserved. Also there is an afunctional and difficultly formalizable requirement to minimize changes in schema of the database and in signatures of stored procedures. This requirement was introduced by the customer to decrease the cost of the following adaptation for the new DB of client applications.

The project is organized as simultaneous progress in two ways: analytics of the initial DB and development of the set of tools for automatic migration. During the project DB migration process improves continuously to entirely automatic migration. For this purpose a set of tools is developed for automation of all predesigned steps: database schema transformation, migration of stored procedures and data transfer. At the same time a complex system for quality assessment is being developed.

Absence of the documentation, which describes functional behavior, was decided to be compensated by an automation of comparison between behaviors of the initial DB and the migrated one in functional scenarios that correlate with business use-cases. Functional complexity of the system (2.5 million lines of stored code) and huge amount of data (6 Tb) considerably complicates organization of the QA process. Additional changes made in schema and in stored code during migration also complicate the automation of the comparison.

II. RELATED WORKS

Methodologies for database migrations are described in several papers. In the article [2] the example of data migration methodology is presented. Migration process between DB with different data models, its risks and problems are described in [3] and [4]. Methodologies for legacy system migration are presented in [5].

Database migration projects have special risks and problems. Authors of [6] propose the data migration triangle for project management in this area. One of dimensions addresses quality assurance. Typical risks, testing and QA techniques are described in [7]. Testing during a database migration lifecycle is considered in [8].

Differential testing was initially proposed by McKeeman [9]. It is a special case of random testing to detect differences between different implementations. Regression testing is discussed in [10] where differential unit-tests are proposed for detecting differences between versions of the same unit. In paper [11] a tool which can identify the cause of regressions by trace analyzing is discussed.

There are a lot of works about data validation. The paper [12] describes it with emphasis on automation, quality and security of data validation process.

An experience of the migration testing can be found at [1].

III. QA PURPOSES

The main purpose of the migration project is to result in a new DB, which contains the same data and has functional correspondence with the initial one. Testing, especially the functional one, is necessary for inspection of migrated DB.

The initial DB is accepted as a model for the migrated DB by formulation of the problem. Behavior of the initial DB is considered as correct and the migrated DB must functionally correspond the initial one within the accuracy of documented changes, which contain renames, changes in a

database schema and consistent changes in procedures semantics.

Goals of the functional testing are:

- Verification of data migration completeness
- Functional correspondence between the initial DB and the migrated one.

Migration tool developing process also needs a control and permanent verification is needed for results of its work. So, objects for testing are:

- Code of the migration tool
- Data migration procedure
- Code for RDBMS Oracle

Functional testing must solve the following problems:

- To provide constant control for correspondence between generated code Oracle PL/SQL and projections designed in the migration tool
- To provide constant control for regression during the migration tool development
- To provide control for functional correspondence between initial (SQL Server) and migrated (Oracle) DBs
- To provide control for data migration correctness
- To provide everyday control for code migration completeness

The QA is executed in two main directions: generated code testing and the data migration process testing.

IV. MIGRATED CODE TESTING

A. Testing of Migratred Code Functionality

Functional testing process is based on a synchronous playing of prepared traces in two DBs (initial and migrated). Trace is a sequence of queries to the DB, which formalizes an interaction between the DB and client applications. The main requirement to the set of traces is sufficient functional coverage. It is a black-box technique, when only external effects are checked. They include output parameters, result-sets, changes inside DB etc.

First traces were created from testing scenarios which were given by the consumer. Traces were collected from industrial servers in order to get more precise information about functionality used in the maintenance. This helps to enlarge the trace set with more priority scenarios which cover more important functionality. Additional synthetic traces were also developed for testing of rarely used functionality.

Using only this kind of testing isn't convenient for the specific project. The whole migration process lasts several hours and it's too long to wait for results of small changes. So some kinds of errors (e.g. incorrect construction transformation) ought to be detected at earlier stages.

B. Early Determination of Defects

Primary migration tool testing is based on small tests, which cover main functionality. These tests are designed for early regress determination, so among them there are examples for all code constructions. A test set is executed automatically after every commit in a version control system and allows prompt detection of an incorrect construction transformation. Analytics and developers increase number of tests during the development of the tool.

In addition, automatic loading and compilation in Oracle are executed every day on procedures translated with the most recent version of the migration tool. They allow to control a number of correctly (syntactic correctness) translated procedures, and show errors appeared in code. This basic testing is especially urgent during the active development of the migration tool.

Testing based on procedures compilation isn't enough for providing syntactic correctness of the code in the specific project. A lot of procedures which contain critically significant functionality use dynamically generated queries. Additional functionality was developed for dynamic SQL, which allows a detection of statically (without procedure execution) lines of literals, which generate incorrect dynamic query for sure.

C. Testing by Trace Playing

The trace recording method is based on MS SQL Server 2005 embedded tools. Queries are captured and saved during the using of DB by customers. Traces for functional testing are recorded on the initial DB in a single-user mode. Each tester has an individual virtual machine. Virtual machine state is saved with deployed DB before trace recording. After recording traces are converted into a unified view which is based on XML. The unified view represents original structure of the initial trace. It consists of batches splitted in queries. It keeps an original text of each query, which was executed on SQL Server, and a text for Oracle. Queries for Oracle are generated automatically by the trace transformation with the rules which were used in stored procedures code transformation. For this purpose integration with trace transformer was added.

The trace playing always starts from the same saved state, determined by the saved state of the initial DB. The migrated DB is a result of migration process application to the saved state of the initial DB. So the trace playing always starts from two DBs in equivalent states. A synchronous playing is also executed in a single-user mode. It provides determined order of the stored procedures execution and a regular repeatability of the trace playing result. Distortions are observed only in results returned by disordered samplings, but the trace playing tool takes this problem into account during the result analysis.

Traces are played with a special developed utility. Result sets received on each step are being compared as difference sets A-B and B-A. If both of difference sets are empty then the result is accepted and translated code is considered to be functionally equivalent to the initial T-SQL one. Otherwise, an attempt to compare sets A-B and B-A is performed by rows and after that by columns for the error localization. Results of comparing are logged into a report. Each trace has

its own record. Errors which were found during the code execution are reported too.

For the QA improvement another monitoring was added which strengthens control on the equivalence of initial and migrated DBs. It looks for changes made during the trace playing inside DBs. For this purpose triggers on events INSERT/UPDATE/DELETE were added for every table in initial and migrated DBs. Triggers write down information about all changes made during the trace playing into a special audit table.

For each section of the trace triggers write down information about the fact of execution an operation on data. It contains a table name, an operation type (INSERT/UPDATE/DELETE), a number of changed values and a hash key of the value collection. This functionality is implemented in one trigger for all three operations on each table. After the trace playing values in the correspondent audit tables are compared. If any difference is found, correspondent tables are also compared. The result of using of this method shows that this testing can find differences in the number of deleted rows which can't be found by comparison of the returning record sets. Noticeable efficiency decreasing wasn't observed after the triggers addition.

D. Control and Providing Testing Coverage Completeness

Testing scenarios which were used for the trace recording were given by the customer. It is supposed that they cover sufficient functionality of the system. For test coverage (completeness) quality assessment source codes of the stored procedures were automatically changed on test servers. Logging instructions were added which allow to backtrace a sequence of operators in order they were executed with an acceptable accuracy. At the beginning of each line code section a command is added which inserts information about passing through a checkpoint into a special table. During the trace playing on the test server a log table is generated. With knowledge about all checkpoints and their places it's possible to count code test coverage using this log. Synthetic tests are counted separately.

For the testing coverage control traces from industrial server are also used, and it helps to determine how the test set covers wide used functionality. But the customer gave only a few industrial traces.

The most completeness cover could be provided with traces from the industrial server, collected during a long period of time. But there are some problems with their reproduction. Firstly, traces and the DB image must be consistently depersonalized before their transfer to third persons. It makes no difficulties to implement such functionality while having such analysis level of the migration tool, but it's impossible because of the project time limit. Another problem is the recording of the reproduced traces even with solved problem of the consistent depersonalization. Values returned in samples to client applications can depend on current state of the DB, for example IDENTITY column. In certain cases results made during parallel sessions are not comparable. Possible solution of the problem is to substitute IDENTITY generation (and a column type converter) in the initial DB for more suitable functions. But this substitution mustn't have influence on efficiency.

V. DATA MIGRATION TESTING

A DB size is impressive, so there is no guarantee, that functional tests can find an imperfection of data loading. So the QA of data loading procedure implementation is based on return codes, logs analysis of all system utilities used in a load chain, a data integrity control embedded in the DB and an additional control of the data loading result – a validation[2].

The validation makes it possible to assure that all data reloaded successfully to the new DB after all necessary documented transformations. This DB is a key component for a valuable part of customer business, so the validation is very urgent after the DB migration of such size. The database schema contains more than 2000 tables and at almost 10000 columns, some tables contain tens of millions of records. Foreign keys as an instrument of data integrity practically aren't used.

The validation process must check not only objects content but the whole database schema verifying existence and state of objects. Full validation for checking DBs equivalence with such volume DBs needs a huge amount of resources, especially a time resource. For a regular process another method is needed. It must require much less resources but have a good result confidence.

At first, validation by row counting was used for primary testing. Measure of success was a table's row number coincidence in initial and migrated DBs. Implementation was pretty simple. It was necessary only to count a number of rows in all tables and to compare results. Script was automatically generated during the database schema and stored code transformation. One of method's advantages is its speed. It has high speed, especially on tables which have primary or unique keys. On this tables number of rows is counted by index and full scan of the table is not performed. Even such simple method revealed unsuccessfully migrated objects. But this method doesn't verify objects values and this is a big disadvantage, especially in migration which is accompanied by the data type transformation.

For the validation result reliability improvement another method was implemented. It is based on hash keys comparison. In the initial DB hash keys are calculated for all columns in every table and results are saved in a separate table. Procedure for hash keys calculating have the following requirement: values of hash keys must be independent from table strings traversal order, because the rows order in the sample can be different and ordering is a very slow operation, it can strongly reduce the speed of the validation. At the current implementation the XOR operation is used. It is commutative and it is embedded into SQL. However research is being made in order to find another function which makes more qualitative hashing but at the same time is very fast. Calculation code is implemented in T-SQL. This code and table with hash keys are being migrated with database schema and stored procedures. Migrated hash calculation procedure on migrated data must give the same

value as an initial one on the initial DB. This also makes an additional testing of the migration tool.

VI. Results

Described testing strategy was implemented within the project. The unit test set contains a hundred of tests for different input language constructs. Permanent control over transformations of the constructions has helped to save a vast amount of men-hours. Compilation in Oracle has often showed a regress made in the previous day, so it wasn't difficult to isolate faults. Both controls have taken just several minutes which is nothing in comparison with migration process.

Trace comparison has been conducted with more than 400 traces and is still growing. Trace playing process takes 6 hours. As a result of the discovered differences investigation a lot of problems were found, and some of them forced to improve or change introduced projections. Moreover, this testing methodology has discovered problems in the initial DB (e.g. some queries return first element from unstable unordered selection, so the result can't be assured).

Evaluation of the testing coverage showed that testing scenarios which were given by the customer had covered about 14% of operators. Additional synthetic scenarios made it possible to increase this value to 33%. During the functional testing about 49% of procedures were executed. Moreover, a huge amount of dead code (more than 40% of operators) was found in the initial database, so the resulting coverage is enough. 20% were confirmed by the customer as acceptable test coverage. This number is based on previous experience in reengineering and correlates with Pareto's principle.

Data validation process has had two implementations. First implementation (by line counting) was fast, and it had found several losses during the data migration at early stages of the project. Next implementation based on hash keys comparison has helped to improve data migration process and it can provide rather high probability of the data migration correctness. Validation process has taken 6 hours on test servers which is much less than full validation.

VII. Conclusion

This paper presents an experience of the QA organization in a technically complex project of DB migration. Described methods were implemented and tested in practice and show their efficiency.

In spite of positive results of the current QA organization and automation some methods can be improved. The main direction of methodology improvement is supposed to implement trace recording and playing from the industrial server. In order to achieve this synchronization problem and depersonalization problem are needed to be solved.

References

[1] Sneed H.M.,"Selective Regression Testing of a Host to DotNet Migration", Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference J. ds, 1892, pp.68–73.

[2] Hudicka J., "The Complete Data Migration Methodology", Dulcian Inc., June 2000

[3] Chang-Yang Lin, "Migrating to Relational Systems: Problems, Methods, and Strategies", Contemporary Management Research, Pages 369-380, Vol. 4, No. 4, December 2008

[4] Maatuk A., "Migrating Relational Databases into Object-Based and XML Databases", Doctoral thesis, Northumbria University, 2009

[5] Wu B., Lawless D., Bisbal J., Grimson J., Wade V., O'Sullivan D., Richardson R., "Legacy System Migration : A Legacy Data Migration Engine", Proccedings of the 17th International Database Conference, October, 1997. pp 129-138

[6] Klaus Haller, "Data Migration Project Management and Standard Software – Experiences in Avaloq Implementation Projects", Proceedings of the DW2008 Conference, St. Gallen, Switzerland, 2008

[7] Matthes F., Schulz C., Haller K., "Testing & quality assurance in data migration projects", Software Maintenance (ICSM), 2011 27th IEEE International Conference.

[8] Patil S., Royy S., Augustinez J., Redlichx A., Lodha S., Vin H., Deshpande A., Gharote M., Mehrotrak A., "Minimizing Testing Overheads in Database Migration Lifecycle", The 16th International Conference on Management of Data (COMAD), 2010

[9] McKeeman W., "Differential testing for software", Digital Technical Journal, 1998

[10] Elbaum S., Chin H., Dwyer M., Dokulil J., "Carving differential unit test cases from system test cases", FSE'06, 2006.

[11] Hoffman K., Eugster P., Jagannathan S., "Semantics-aware trace analysis", PLDI'09, 2009

[12] Manjunath T., Ravindra S.,Mohan H., "Automated Data Validation for Data Migration Security", International Journal of Computer Applications (0975 – 8887)Volume 30– No.6, September 2011

[13] Microsoft Developer Network Library, http://msdn.microsoft.com/en-us/library/default.aspx

[14] Oracle Documentation, http://www.oracle.com/pls/db112/homepage