Ilkogretim Online - Elementary Education Online, 2020; Vol 19 (Issue 4): pp. 8599-8603
http://ilkogretim-online.org
doi: 10.17051/ilkonline.2020.04.765250

Challenges During Conversion Of Cobol To Oracle Database In Multan Electric Power Company

Sajjad Hussain Qureshi Department of Computer Science, Govt. College University, Faisalabad, Pakistan.

Muhammad Danial Jan Department of Computer Science, Govt. College University, Faisalabad, Pakistan.

Muhammad Sohaib Akram Department of Computer Science, Govt. College Mailsi, Vehari, Pakistan.

*Corresponding author: Sajjad Hussain Qureshi, Email address: sajjads2002@yahoo.com

Abstract

The Multan Electric Power Company (MEPCO) was using open VMS operating system to run high level language (COBOL) for its business purpose. The operating system is command based which was needed to replace with the graphical user interphase. The data warehouse was converted from sequential, indexed and relative file organization to an Oracle relational file system. The special programs were written to translate COBOL to Oracle database. This shifting of database was necessary to support centralization of data, reducing operational cost and to adopt latest advancement in the technology. It turned out the MEPCO as one of the largest company getting benefited from modern application softwares. This manuscript describes how to test the functional equivalency of updated and old applications, as well as the main testing obstacles and the lesson learned along the course of running and maintaining the updated application. Functionally equally working systems were developed but experience staff, parallel and vertical software feature development, staff trainings, troubleshooting, restriction on database and real time updation of information were always challenging.

Keyword: COBOL, Oracle, troubleshooting.

1. Introduction

Multan Electric Power Company (MEPCO) is one of the main distribution company in the Pakistan which deals with selling of electricity. Its IT department is responsible for accurate billing on the basis of unit sold provided by field formations. Currently, decentralized system using COBOL is operated which needs to replace with relational database model having tables to store the data. Thousands of lines in COBOL code are now part of the application, which is used for important business tasks like billing, cash posting, basic data entry, meter change information, MIS reports etc. The system is developed by many experienced programmers but it was unable to reduce the operating cost. Moreover, latest recruited staff has no knowledge about COBOL due to eliminating of COBOL subject from the institutions.

The use of client/server environment in small needed application urges the need of shifting of system to latest relational database (Fowler, 2012). It not only minimizes the company expenses but also gives chance to new computer engineers to explore their hidden abilities. The plan was to use code and data translation to move the programme from the old system rather than developing a new application. This method starts with writing suitable programs to integrate data from old to new database. The development of application from scratch is more costly and need more time and knowledge about the existing system working (Sneed & Verhoef, 2013). Moreover, consumer data is more important and is necessary to maintain its record. The programms to translate data and code was supplied by the Pakistan Information Technology Company (PITC). Inhouse development costs more than vendor (Tiwana & Kim, 2016). The majority of new software developers lack the understanding of VMS operating system concepts and COBOL programming languages, which is also be necessary during deployment. As the members of the project team, we report on the successes, lesson learned and failures of the system transformation. Legacy code translation is known to present certain difficulties (Trudel, 2013). However, application testing proved to be the most challenging and time-consuming (Anand et al., 2013).

Unexpected obstacles during testing resulted in considerable delays. To evaluate functional equivalency between the modern application and the old application, a comprehensive testing procedure was needed to develop (Cadar et al., 2008). The main objective of the study are 1) to sort out the challenges encountered during application's testing, 2) gap between old and new softwares to handle customer data and flaws in the testing procedure.

2. Material and methods

During transformation, first step was to take backup of old data and use the old system to gather inventory. Breakdown of the working procedure into different small module is very important to get more reliable results. This is performed in second step. In third step, we updated the database design. Now transformation of data from old to new database was performed which was termed as data migration stage. In this step, code already written was used to convert the sequential data into relational database. After this, date verification from both old and new system was performed considering each record. When all kind of shifting was completed, the training on new system was given to the users to work on new system. This phase also involved to check the acceptance level of the users.

The vendor offered a runtime framework that implemented various mainframe services and utilities in addition to translating code and data. This made it possible for the compiled code to function on the contemporary platform and still communicate with its surroundings in a manner that was reminiscent of the mainframe. Replacement parts were created in the following situations: when the runtime framework is unable to support legacy application dependencies. when a commercial software product is not a viable option; or when it makes sense to utilise the capabilities of the updated environment.

We were able to test functional equivalency for the full application after testing it at the component group level initially, and then progressively working our way up to that level after implementation. Since these environments can't be tested on individual developer PCs, we may design and test code to automate configuration, full batch scheduling, and infrastructure-related performance enhancements. The development environment used static test data, while the staging environment used dynamic test data from the current day. Every day, the batch process was conducted in the old system test area to generate the test data for the current day, and the output files were subsequently sent. This allowed for the testing of scenarios that varied based on the day of the week or month, as well as the verification that batch output from one day was handled appropriately the next day.

3. Results

Most of the project's time is spent in testing. These difficulties were the main reason for late completion of the project. Nearly all older application components lacked sufficient test coverage to provide the data needed for functional equivalency testing. Due to this, a great deal of effort was placed into attempting to examine the inputs, outputs, and occasionally even the internal behavior of components when an investigation of the underlying cause of test failures was necessary. This was especially challenging and time-consuming for batch jobs that required additional complexity. To verify the batch process's functional equivalency from beginning to end, proper controls were set. The legacy application has amassed a considerable quantity of legacy code. We had to take our time locating and eliminating this code because it had not been maintained throughout the years. Many transfers used unsecured protocols, connections were configured in a variety of ways, and we were unaware of all the consumers. Delays in consumer discovery, secure protocol upgrades, and configuration management coordination were substantial.

Display fields and computation fields coexist in many files. This kind of file had to be transferred since it was necessary to evaluate the compiled routines' functional equivalency. The arithmetic fields must stay binary encoded throughout the transfer of these files from the old system. The complexity increases while handling the various versions of each file and their vast volume. New tools have had to be created in order to examine, alter, and compare files with fixed or variable block structures and computational domains.

We mapped each layout to a separate table and improved the maintainability. This increased performance while mapping to a single table. When converted to a single relational database table using the outcome of the cursor mapping, frequently entails moving the cursor to an indexed location using a key, and then repeatedly moving the cursor forward or backward to the next or previous record until some condition is met. This was extremely slow because each of the tables' result sets required open cursors to be maintained, and the data access logic had to determine which cursor contained the previous or next entry.

During the testing of a few batch jobs, we ran into performance problems that were not fixed by the previously mentioned database redesign. Without changing application code, middleware can access and modify data using external configuration parameters. The in-memory data structure is operated on once all required data has been put into memory and modifications are written back at the conclusion of processing. Network latency and database I/O bottlenecks were removed by executing these actions in memory, but we were unable to run further batch jobs concurrently if they relied on the same database tables. There were weaknesses in the testing procedure even with the significant investment in testing. We ran across vulnerabilities in production after finishing the user acceptability testing.

4. Discussions

The majority of modernization initiatives are completed later than anticipated "as a direct result of misunderstanding of legacy applications (Zacharewicz et al., 2017). As a prelude to this project, we may have improved the application's initial state while gaining this expertise. This probably lessened the problems indicated and provided guidance for the ensuing planning, estimating, and project management (Belassi & Tukel, 996).

To estimate when comparable future work will be completed, we explored dividing the work into smaller parts and timing the completion of each job. Expectations of an early delivery were pushed back by testing obstacles relating to batch processing because later work packages were more batch-oriented than earlier ones (Pufahl & Weske, 2017). When we first organized the work packages, we may have had a better grasp of the application, which may have prevented this problem from coming to light too late.

Our aim of offering a contemporary application that is functionally equal to the original application has been accomplished. Functional equivalency has been demonstrated by means of an intricate testing procedure. Unexpected testing issues caused it to be supplied later than anticipated, but at a significantly cheaper cost. The current application has shown to be fairly steady in production, despite a few production-related problems, and bills are still distributed each day. However, creating new features is still challenging. It demands information that the software engineers in the team do not have: knowledge of VMS operating system and COBOL programming identifiers. The team's COBOL programmers were not proficient in Oracle, but they are familiar with the language and concepts. Since we are developers with several trainings, we hope that this aids in knowledge transmission. We also observe that the lack of enthusiasm for working with compiled code made it challenging to get fresh software engineers.

References

Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B., Grieskamp, W., ... & Zhu, H. (2013). An orchestrated survey of methodologies for automated software test case generation. Journal of systems and software, 86(8), 1978-2001.

Belassi, W., & Tukel, O. I. (1996). A new framework for determining critical success/failure factors in projects. International journal of project management, 14(3), 141-151.

Cadar, C., Dunbar, D., & Engler, D. R. (2008, December). Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In OSDI (Vol. 8, pp. 209-224).

Fowler, M. (2012). Patterns of enterprise application architecture. Addison-Wesley.

Pufahl, L., & Weske, M. (2017). Requirements framework for batch processing in business processes. In Enterprise, Business-Process and Information Systems Modeling: 18th International Conference, BPMDS 2017, 22nd International Conference, EMMSAD 2017, Held at CAiSE 2017, Essen, Germany, June 12-13, 2017, Proceedings 18 (pp. 85-100). Springer International Publishing.

Sneed, H. M., & Verhoef, C. (2013, September). Migrating to service-oriented systems (Why and how to avoid developing customized software applications from scratch). In 2013 15th IEEE International Symposium on Web Systems Evolution (WSE) (pp. 91-96). IEEE.

Tiwana, A., & Kim, S. K. (2016). Concurrent IT sourcing: Mechanisms and contingent advantages. Journal of Management Information Systems, 33(1), 101-138.

Trudel, M. (2013). Automatic translation and object-oriented reengineering of legacy code (Doctoral dissertation, ETH Zurich).

Zacharewicz, G., Diallo, S., Ducq, Y., Agostinho, C., Jardim-Goncalves, R., Bazoun, H., ... & Doumeingts, G. (2017). Model-based approaches for interoperability of next generation enterprise information systems: state of the art and future challenges. Information Systems and e-Business Management, 15, 229-256.