Test data distribution system for OSS/BSS-systems testing

Steven Nikolaev[1,*] and *Igor* Sitnikov[1]

[1]IRIT-RTF; Ural Federal University named after the first President of Russia B.N. Yeltsin, Yekaterinburg, Russian Federation

Abstract. OSS/BSS-systems used by telecommunication companies to conduct their business are a combination of several subsystems that closely interact with each other. Since the activities of telecommunications companies are related to the processing of personal data of individuals and legal entities, such systems should be subjected to testing, which should continue until the end of the life cycle. The huge size of telecommunications companies' databases, as well as the number of tests required to cover the entire functionality of OSS/BSS-systems are an obstacle to conducting rapid testing. This article describes an approach to testing organization that solves this problem, based on the selection and keeping the test data up to date. The effectiveness of the proposed approach has been demonstrated experimentally.

1 Introduction

OSS/BSS systems are the fundamental set of software aimed to support internal business processes of telecommunication companies. They are responsible for the interaction between all information systems providing communication services, managing internal processes, committing billing operations and controlling company resources [1].

OSS/BSS systems that being used by every telecom operator are designed to perform the same tasks and are based on generally accepted principles and standards. However, the implementation of OSS/BSS systems may vary depending on the type of services and scale of the business itself.

OSS is stands for Operational Support System. It is the class of software responsible for automating the management of network and its equipment. The entire technical aspect of the activities of telecommunications companies is concentrated here.

BSS is stands for Business Support System. BSS includes software that provides financial management, accounting, personnel, projects and fixed assets management. BSS systems consist of three main subsystems:

1. Billing systems;
2. CRM;
3. ERP.

* Corresponding author: i.o.sitnikov@urfu.ru

Billing systems are systems providing a set of processes and solutions that organizes information collection about the use of services, accounting and payment processing. Billing systems are a particularly important component for commercial telecommunications companies, regardless of their type because companies can keep an accurate record of the total profits from the provision of communication services to its customers, as well as provide themselves with an additional module to protect against fraud. There are various solutions for billing systems developed by huge amount of organizations including such large companies like Microsoft (Dynamics 365) and PayPal (PayPal Invoicing). However, it is quite common when telecommunications companies develop a billing system on their own, taking into account the existing infrastructure.

Customer Relationship Management or CRM is a system aimed to ease interaction between company and its clients with maximum performance. They are affecting many channels of interaction: personal service in retail branches, telephone, email, chat, advertising, etc. All registered calls and operations are stored in a single repository, and then can be used by a telecommunications company for analysis, in order to improve the efficiency of the services provided, as well as customer service channels.

ERP systems are responsible for enterprise resource planning: labor, human and financial resources management, asset management, etc.

Company's evolution in the field of communication is accompanied by huge losses and many unforeseen difficulties, due to the scale of this type of business. The high cost of any internal and external reorganization can be explained by the need of sustaining continuous providing with services and even the most minor incident can lead to the loss of a huge part of the clientele. This is especially noticeable for leading telecommunications companies. As for difficulties, we can say that nowadays the telecommunications services market has reached a new stage: it is become more about keeping clients rather than obtaining new ones because cellular communications and mobile devices have become extremely accessible and used by almost every citizen of developed country. Such change of business vector can be seen in a quantity of new services and their costs, not to mention that companies are always ready to give special offers to clients that trying to change their current operator with a new one. To sustain such fast reaction to changes in business and creating new products any company must be sure in stability of their services. In other words, telecom operators have to pay attention to quality of their products and services that can be achieved only after thorough and complex testing.

2 BSS/OSS-systems testing

2.1 Description of BSS/OSS-systems testing process

There are two main approaches to testing:

1. Manual testing;
2. Automated testing.

Manual testing is a type of testing in which software is examined for errors and defects directly by a person. The main idea of manual testing is that the tester is directly involved in the testing process, without using any additional software. This simulates the interaction of the end user with the current version of the product.

Advantages of manual testing:

1. The quality of testing is determined only by the curiosity and imagination of the tester [2];
2. Low threshold for entering the testing process.

Disadvantages of manual testing:

1. Requires constant involvement of the tester in the process;
2. Monotony of testing process;
3. Complex systems require many manual testers to cover all current and new features in a timely manner.

Manual testing does not require a specific qualification from the tester and the use of various additional hardware or software. It is enough to have a working machine with access to the systems under test and the most basic interfaces needed for interaction with them. However, testing of the same type of objects (for example, testing of a new tariff line) is associated with the implementation of repeated actions that could be delegated to a computer. Therefore, as an alternative to manual testing, automation of such repeatable steps drastically raised its popularity.

Automated testing is a type of testing in which software being explored for errors and defects with the partial participation of a tester. Most of the testing process is being performed by an auto-test - a special program code that interacts with the objects of testing and generates a report on the test result. The main task of tester is to write auto-tests, control their execution and analyze their results. At the same time, it is worth noting that a special system (for example, Jenkins CI server [1]) can be responsible for the execution of auto-tests that can help to configure special builds helping to run auto-tests to run on a schedule or at a specific events.

Advantages of automated testing:

1. Automate the reproduction of the same actions with minimal involvement of the tester;
2. Does not require a large number of testers;
3. Reducing the time of testing;
4. A well-written auto-test is deprived of the human factor and is able to produce the most reliable results.

Disadvantages of automated testing:

1. Writing an auto-test requires a certain degree of qualification from the tester;
2. Scripts require constant support, in view of the expected ongoing updates of OSS/BSS-systems;
3. Auto-test with a logical error can produce inadequate results.

Automated testing requires much more resources to prepare for writing and executing tests. The tester is required to have knowledge of at least one of the popular high-level programming languages (for example, Java or Python), ability to work with UNIX-like operating systems, basic knowledge of networking, information security, version control systems, CI/CD principles, knowledge of databases, etc. At the same time, test automation team is only a small part of the whole automated testing process – it is necessary to create a specific testing environment and select people responsible for the subsystems as consultants. Despite such high demands, automated testing can significantly improve the quality of products and services produced by minimizing the human factor in the testing process while simultaneously accelerating it.

2.2 Problems of BSS/OSS-systems testing

One of the typical problems of testing OSS/BSS systems is the large volume of databases. The higher the client base of a telecommunications company, the higher the amount of data to work with. It should be noted that not all records from the table could be useful for a particular test. Testers have to form relatively complex queries in the database, providing data filtering at the very start of testing process. This is necessary in order to reduce the complexity and remove redundant logic from the functions and methods that work with the results of these queries. The huge size of the database and the complexity of the queries

lead to the fact that the auto-tests will spend a large amount of time on the selection of input data.

One of many possible ways to solve this problem is to reduce the size of test databases. This can occur at the deployment stage of the test environment. When replicating databases and devaluing the data contained in it, you can control the number of records that must be transferred as a result. However, there are two disadvantages to this method:

1. Reducing the volume of databases reduces reliability of the results of testing process;
2. The corruption of data that occurs when certain tests were started simultaneously.

Obviously, full replication of production database is too expensive so companies will run certain procedures that will reduce the volume of test databases, but the amount of data must be enough to be used by multiple executions of auto-tests without the need of recreating test database very often. That means that the volume of databases can be still considered a problem.

In addition to the problem of the size of the databases, there is also a question about the distribution of data between tests. Let us assume a situation in which two different tests that select input data by the same criterion were run simultaneously. In this case, the result of both tests can be extremely unexpected. Therefore, it is necessary to establish a specific mechanism for delegating test data between auto tests.

3 Test data collection and distribution system

To solve the problems that were described in the previous section, it is proposed to develop a special system for the preparation of test data, which is an intermediate layer between the test environment and the database containing necessary information. This system must consist of two main functions:

1. Selection of test data according to certain criteria and their storage in its own database;
2. The distribution of access to this data between tests.

Selection of test data occurs after the first run of tests. All requests to receive information from the database are being sent through the system. At the initial tests execution the system receives and remembers the request, obtain data for this request from the test database, returns the desired value to the test, and then replenishes its own repository with a new set of test data in the background. When you run the tests again, the system will immediately return information from its own repository, which will reduce the tests runtime.

The distribution of data access between tests is organized by a special mechanism, which will be triggered whenever tests are requesting data. The mechanism sets a lock flag on the record, which will reset after some short time has elapsed.

Based on the aforementioned information, let's formulate the main components that should be included in the test data storage system:

1. Database storing collected test data and info about selection criteria;
2. The mechanism for updating the test data;
3. The mechanism of blocking data that is currently processed by tests;
4. Interface of test interaction with the system.

Let's also distinguish additional functionality:

1. Logging, which includes records of requests for data, the replenishment of the system with new records and other actions, marked by a time stamp;
2. Graphical user interface through which you one can manually control the system.

The closest analogue of this system is the master-server used in the Google File System developed by Google Inc. [4]. The master-server in GFS is responsible for preparing the

metadata in advance to identify the desired server with the data of interest to the user. This approach helped Google to significantly reduce the time needed for processing search queries with minimal costs for equipment.

4 Experiments and analysis

4.1 Test scenario and test environment

A remote server rented from the cloud infrastructure provider DigitalOcean have been used as a test server. Characteristics of remote server are presented in Table 1.

Table 1. Characteristics of remote server

Parameter	Value
Operation System	Ubuntu 18.04
CPU	1x1GHz
RAM	1 GB
SDD	25 GB

The process of connecting additional package of services to a subscriber was chosen as a test scenario. The scenario consists of six steps:

1. Test receives the identifier of the package as input data;
2. Test makes a request to the database to receive a suitable subscriber who does not have this package;
3. Test transmits obtained identifiers of the subscriber and the package to the system, which is responsible for connecting packages and services,;
4. The system modifies the data received by the subscriber and connects the package;
5. The system returns the result of the connection;
6. The test analyzes the result and displays the corresponding message.

The scheme of the test scenario is presented in Figure 1.

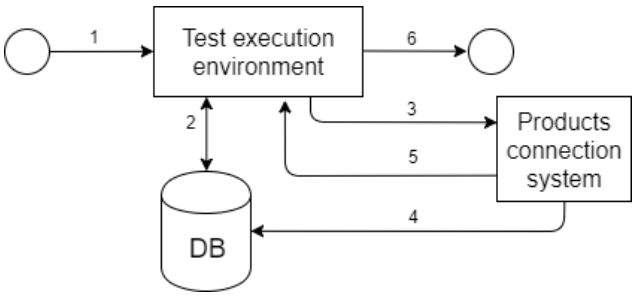

Figure 1. The scheme of the test scenario.

In accordance with the test scenario, to implement the simulation environment, it is necessary to prepare a database and create a product connection server.

PostgreSQL was chosen as the DBMS, since it is multiplatform and being distributed under the PostgreSQL License, similar to BSD and MIT licenses.

The final database includes six tables:

1. CLIENTS - table with clients;

2. SUBSCRIBERS - table with subscribers. Have index on the field "subs_id";
3. RATE_PLANS - table with rate plans;
4. PACKS - table with packages. The rate plan is a set of different packages giving a specific type of service each;
5. RTPL_PACKS - a table connecting packages with rate plans. This table stores information about which packages will be connected to the subscriber on connection of a particular rate plan, as well as optional packages that can be connected in addition to the rate plan;
6. SUBS_PACKS - a table connecting packages with subscribers. This table stores information about which packages are already connected to the subscriber.

The ER-diagram of the developed database is presented in Figure 2.

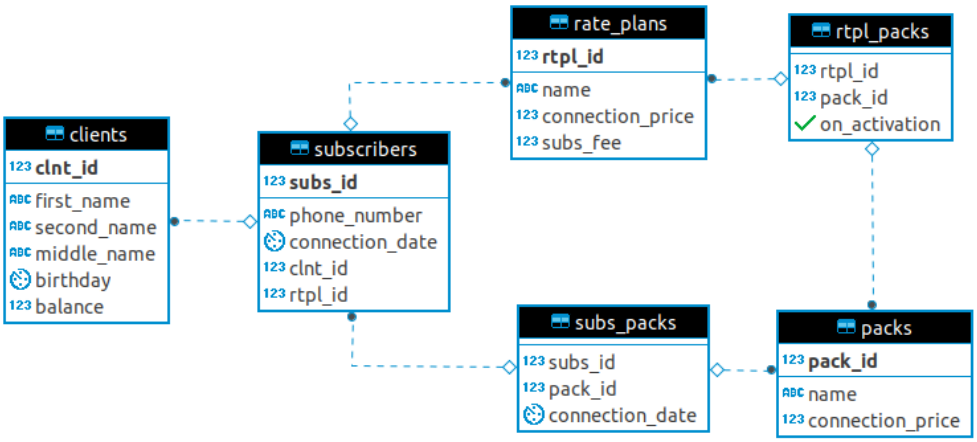

Figure 2. ER-diagram of the database.

Creation of the database was automated by additional scripts including SQL-queries and modules written in Python. All clients' names are random sets of English characters. Date of birth is also chosen randomly, considering that the client cannot be younger than 10 years. The generation of the connection date also takes into account various impossible situations. For example it is impossible for a client to connect a rate plan years before the client was born. As for packages, when it comes to generating records in the "RTPL_PACKS" table, 10 random optional packages were selected for each tariff plan. The total numbers of records in the resulting database are shown in Table 2.

Table 2. Rows quantity in tables

Table	Rows quantity
CLIENTS	1 million
SUBSCRIBERS	1 million
RATE_PLANS	12
PACKS	22
RTPL_PACKS	120
SUBS_PACKS	4666700

Based on the description of the tables, the package is connected to the subscriber only when there is a corresponding entry in the "SUBS_PACKS" table. This table is modified via using a products connection system.

To develop a system for connecting products and services, Python 3.7.0 was used, as well as the «Flask» microframework, which was used to develop a small API for interacting with the system. The module «psycopg2» was used to interact with PostgreSQL. This module can be easily installed via the «pip» installer supplied with Python on any platform and does not require additional installation of PostgreSQL compatibility drivers. The choice of these tools is largely associated with ease of use and multiplatform nature.

Product connection system consists of two functions that necessary for current test environment:

1. Processing the incoming POST-request for connecting packages to subscribers;;
2. Sending request to database in order to modify «SUBS_PACKS» using given information.

The standard web server of the Flask framework is used as a server, but it is not designed to work in large industrial systems. Nevertheless, its functionality is enough to conduct a test scenario simulation.

After preparing all the main components, a small test script was written in Python that executes test scenario and collect results.

4.2 Implementing test data distribution system into testing environment

Let's consider the execution of given test scenario using our test data distribution system. The scheme of the test scenario with new component is shown in Figure 3.

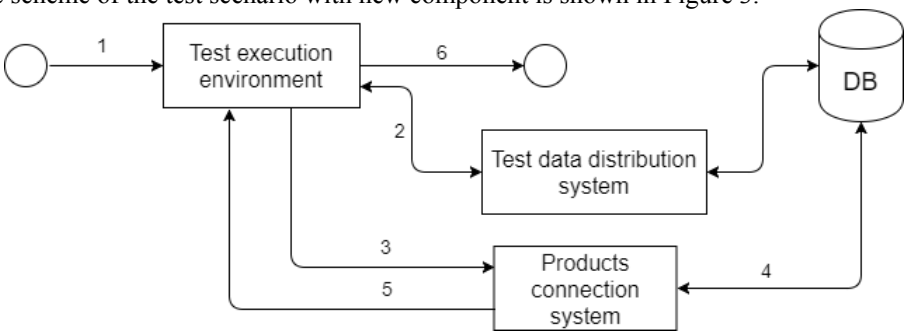

Figure 3. The scheme of the test scenario with test data distribution system.

Test data distribution system was embedded into modeled test environment. Auto-test responsible for recreating given test scenario was modified to send its requests into test data distribution system instead of sending them to database directly.

The system itself was developed on the AIOHTTP framework for Python because this framework offers asynchronous processing of requests incoming from tests that reduces the time needed for processing and sustain effective interaction with test data. PostgreSQL is used as storage, but in the future, we will try to implement Redis instead for a faster interaction.

For described test scenario, the following functionality was implemented in test data distribution system:

1. Interaction interface between auto-test and this system;
2. Small and simple database for test data storage;
3. Database background filling mechanism;
4. Test data blocking mechanism.

Algorithm of interaction between auto-test and the system:

1. Auto-test sends an HTTP-request to test data distribution system, that contains SQL-query to database;

2. The system checks the availability of data that fit to the given SQL-query in its own storage;
3. If the data was found then they are sent back to auto-test;
4. Otherwise, the system saves SQL-query in its own storage, run it directly on the main database and return result of this request to auto-test. Database background filling mechanism will use SQL-request remembered earlier to prepare new set of test data for future tests executions.

Giving data to auto-test triggers test data blocking mechanism. After auto-test connects to the system and obtains needed test data, the identifier of the given data will be memorized. Later on, this identifier transfers to a separate task, which locks the record containing identifier for a certain time.

4.3 Analysis and results comparison

The following SQL-query is executed to get the subscriber ID:

```
select subs_id
  from subscribers
 where rtpl_id in (select rtpl_id
                     from rtpl_packs
                    where pack_id = {pack_id})
   and subs_id not in (select subs_id
                         from subs_packs
                        where pack_id = {pack_id}
                        group by subs_id)
 group by subs_id
```

Execution of this script without implementing the test data distribution system with full output of results on a workstation with same characteristics takes a little more than a day to complete, which is completely unsuitable for testing. Therefore, the test function responsible for sending this request was modified to receive only 200 entries. Request execution time with modifications decreased to 11 seconds.

The first run of the auto-test using the test data distribution system on the same workstation takes the same 11 seconds, because the system is not yet filled with test data and does not know what data should be collected to replenish its storage. When the test was restarted, the runtime was reduced to 0.01 seconds. Thus, performing an auto-test using a test data distribution system has significantly reduced the speed of obtaining test data. When trying to simultaneously run several tests that receive data for the same SQL-query, each of the tests receives its own data thanks to test data blocking mechanism.

5 Conclusion

The test data distribution system has significantly reduced the time it takes to complete a test, by reducing input data receiving time. Several simultaneously running test instances obtaining their own input parameters that prevents data corruption, preserves the integrity of data and the accuracy of the test results. However, the current implementation of the system requires some major improvements:

1. Test data blocking mechanism requires additional ways of setting test data blocking time in order to make data distribution more flexible;

2. Data filling mechanism must be optimized to check test data that was recently given to tests instead of running a background task responsible for actualization of all data stored;

3. Test data distribution needs to be universal by creating additional interfaces to work with other DBMS.

Test environment needs certain modifications too: it is necessary to create an artificial load on the existing systems in order to determine the test run time in a more realistic environment. In addition, it is worth adding one more element to this model, which would assume the role of the billing system and would be responsible for changing subscribers' balance.

This work is supported by Act 211 Government of the Russian Federation, contract № 02.A03.21.0006.

References

1. The Definition of OSS/BSS. Available at: https://en.wikipedia.org/ wiki/OSS/BSS (2010)
2. K. Aksyonov, D. Antipin, T. Afanaseva, I. Kalinin, I. Evdokimov, A. Shevchuk, A. Karavaev, U. Chiryshev, E. Talancev. *5th International Young Scientists Conference on Information Technologies, Telecommunications and Control Systems*, ITTCS 2018; Yekaterinburg; Russian Federation; 6-8 December 2018. Code 144071. CEUR Workshop Proceedings Volume **2298**, 7, (2018)
3. Jenkins User Documentation. Available at: https://jenkins.io/doc/ (2017)
4. S. Ghemawat, H. Gobioff, S.T. Leung, *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 20-43, (2003)