The Journal of Systems and Software 113 (2016) 257–274

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Cost-effective strategies for the regression testing of database applications: Case study and lessons learned

Erik Rogstad [a,*], Lionel Briand [b]

[a] Simula Research Laboratory, P.O. Box 134, 1325 Lysaker, Norway
[b] University of Luxembourg, SnT Centre, 4, rue Alphonse Weicker, L-2721 Luxembourg, Luxembourg

ARTICLE INFO

Article history:
Received 8 July 2015
Revised 27 October 2015
Accepted 3 December 2015
Available online 15 December 2015

Keywords:
Regression testing
Database applications
Classification tree modeling

ABSTRACT

Testing and, more specifically, the regression testing of database applications is highly challenging and costly. One can rely on production data or generate synthetic data, for example based on combinatorial techniques or operational profiles. Both approaches have drawbacks and advantages. Automating testing with production data is impractical and combinatorial test suites might not be representative of system operations.

In this paper, based on a large scale case study in a representative development environment, we explore the cost and effectiveness of various approaches and their combination for the regression testing of database applications, based on production data and synthetic data generated through classification tree models of the input domain.

The results confirm that combinatorial test suite specifications bear little relation to test suite specifications derived from the system operational profile. Nevertheless, combinatorial testing strategies are effective, both in terms of the number of regression faults discovered but also, more surprisingly, in terms of the importance of these faults. However, our study also shows that relying solely on synthesized test data derived from test models could lead to important faults slipping to production. Thus, we recommend that testing on production data and combinatorial testing be combined to achieve optimal results.

1. Introduction

In large database applications, constructing synthetic test data is often deemed too time-consuming because of the large number of data dependencies in a typical relational database. Thus, a common strategy when testing database applications is to rely on production data for testing, i.e. data from the system operation. However, available production data may not satisfy all the requirements of the test plan (test specifications), thus forcing the testers to manipulate and/or extend the production data to make it adequate. Both the task of identifying appropriate test data and subsequently manipulating it if it does not exactly fit the needs, is a tedious process for the testers. The use of production data nevertheless remains common practice due to the lack of alternatives.

In order to achieve effective test automation, for example to support regression testing, we would like to rely on test suites conforming to clear test strategies, i.e. devised in a structured and systematic manner, to ensure predictability of test results both in terms of coverage and fault revealing power. Thus, automatically generating synthetic test data becomes necessary. Yet again, when generating test data, a wide range of possibilities opens up. Rather than finding appropriate production data, the challenge shifts to limiting the amount of test data to generate to a proper sized test suite matching the available test budget, while still ensuring predictable quality. Combinatorial testing is an attractive strategy for generating compact n-way test suite specifications (Dustin, 2002; Hedayat et al., 1999), and consequently a natural starting point for test data generation. Classification tree modeling is a common approach to combinatorial test design (Grochtmann and Grimm, 1993), in which the input domain of the system under test is modeled as a classification tree, which in turn is used to generate a combinatorial test suite specification (Lehmann and Wegener, 2000).

However, based on our experience, combinatorial test suite specifications do not necessarily align well with system usage and the benefits in terms of risk reduction resulting from testing are therefore unclear. In other words, when your test model is fairly complex, a compact combinatorial test suite may mostly consist of test cases that are rarely or never executed in the real operation of the system. To ensure that tests match meaningful and representative executions of the system, we can analyze the operational profile of some of the

* Corresponding author. Tel.: +4745600663.
E-mail address: erikrog@simula.no (E. Rogstad).

http://dx.doi.org/10.1016/j.jss.2015.12.003
0164-1212/

functional areas of the system under test, on which basis we can generate test suite specifications directly aligned with system usage. Alternatively, we can use the operational profile to weigh the properties in our combinatorial models (classification trees), which is then used to generate more representative combinatorial test suites.

In this paper we present an empirical investigation, in a real and representative database application development environment, of various strategies for the selection and generation of test data for database applications. We assess production data and generate synthetic test data following various combinatorial strategies based on classification tree models. We focus on system level regression testing and run actual regression tests to compare the fault detection rate of each strategy. Our case study focuses on the following questions:

- What combinatorial test coverage can be expected from production data? It is important to understand the scope of production data if they are to be used for testing, and also help determine whether additional data need to be synthesized for achieving satisfactory testing.
- How effective are synthesized combinatorial regression test suites, based on pair-wise and three-wise coverage, at assessing and eliminating the risks of failure? Combinatorial test suites tend to be compact, as they target model coverage with a small set of test cases. It is then interesting to explore to what extent the compact test suites are representative of the usage of the system being tested. This will determine whether they are able to detect relevant faults, that is faults that are likely to manifest themselves in practice and present significant risks.
- Can operational profile analysis help guide the synthesis of more effective regression tests? One important question is whether we can use an operational profile analysis to weigh model property values, in order to generate more representative combinatorial test suites. As an alternative to combinatorial testing, we must also investigate whether synthesized data based on an operational profile is an effective regression test strategy, or a complement to combinatorial testing.
- How do test strategies compare, in terms of their ability to detect faults, during regression testing and when generated according to: (1) Synthesized combinatorial test data (pair-wise and three-wise), (2) The operational profile of the system, (3) Synthesized pair-wise test data weighed based on the operational profile, and (4) production data. Whether to test on production data or synthesized data often comes down to what is more practical in a given context. However, assessing both alternatives and comparing them is interesting to understand the consequences of choosing one over the other or combining them. Furthermore, it is interesting to investigate whether synthesized data based on the operational profile can detect similar faults as tests derived from production data.

The remainder of the paper is organized as follows: Section 2 provides information about the industrial context of our work, along with the background and motivation for our case study. Section 3 outlines a practical approach to generate test data, based on classification tree models, and the matching of production data against these models. The design of the case study is presented in Section 4, along with empirical results, discussions, and threats to validity. Related work is reported in Section 5, before drawing conclusions in Section 6.

2. Background and motivation

The Norwegian Tax Department maintains several large database systems. For example, the Norwegian tax accounting system, SOFIE, serves more than 3,000 end users (taxation officers) and handle yearly tax revenues of approximately 600 Billion NOK. Common to these systems is that they process large amounts of data, and accordingly the business logic of the systems is organized into batch programs to ensure efficient data processing. Batch programs typically process large sets of input data and run to completion without human intervention. This makes them suitable for automated transaction handling, and the batches are thereby often scheduled to run periodically. As being a part of the Tax Department portfolio, the systems are subject to changes in taxation laws, along with usual maintenance activities, thus causing continuous changes to the batch programs. It is vital for the tax department to avoid releasing faults upon changes and maintain system quality to preserve taxpayers' confidence. Nevertheless, regression faults in the batch programs have been a struggle throughout the years. Further, the batch programs process large amounts of data, spanning a wide variety of possible test cases, which makes manual testing inadequate to support quality assurance in the context of frequent changes.

2.1. Automated regression testing

To address some of the issues mentioned above, we proposed a partly automated regression test procedure and tool (DART) tailored to database applications (Rogstad et al., 2011). It compares executions of a changed version of the program against the original version of the program and identifies deviations, that is differences in the way the database is manipulated between the two executions. In each test execution, the database manipulations are logged according to a specification by the tester indicating the tables and columns to monitor. The database manipulations from each execution are compared across system versions to produce a set of deviations, which indicate either correct changes or regression faults.

The strength of this approach is that it provides the ability to verify the entire set of test data executed by a batch automatically. As an example, lets say we execute a batch running the tax calculation for 10,000 taxpayers, each constituting a test case. Manually verifying 10,000 tests is far beyond what a tester can realistically handle. Therefore, one would have to pick out a small sample to analyze based on qualified guesses whereas the rest of the 10,000 tax calculations would remain unattended and pose substantial risk to the system release. However, with the regression test procedure suggested above for database applications, all the 10,000 tax calculations will automatically be compared against a previous execution to separate the test cases that deviated from the ones that did not.

2.2. Observations regarding test data

Throughout initial phases of applying the proposed methodology for automated regression testing, we made a number of observations regarding test data, that formed the basis for the work presented in this paper. Our case study is further elaborated in Section 4, whereas the remainder of this section will elaborate on the observations motivating it.

2.2.1. Testing using production data

In a system like SOFIE, like in many other database applications, there are vast amounts of data available from the production environment of the system. Consequently, the testing of SOFIE has traditionally been heavily relying on the use of anonymized production data. In practice, the test data is made available by making a copy of the production database for testing purposes and then reusing input files from the production environment when running tests.

This was our starting point for the regression testing of the batch programs. However, initial regression tests on production data indicated significant redundancy in the data and very unpredictable test coverage. Redundancy was visible when conducting the deviation analysis, as many test cases executed the system in a similar way and triggered the same fault, thus causing many redundant deviations to inspect. The unpredictable test coverage became evident when we ran the same regression test (same batch program versions) using