IDEA GROUP PUBLISHING

ITJ2238

1331 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

Regression Testing of Database Applications

RAMZI A. HARATY, Lebanese American University in Beirut, Lebanon
NASHAT MANSOUR, Lebanese American University in Beirut, Lebanon
BASSEL A. DAOU, University of Ottawa, Canada

Database applications features such as Structured Query Language or SQL, exception programming, integrity constraints, and table triggers pose difficulties for maintenance activities; especially for regression testing that follows modifications to database applications. In this work, we address these difficulties and propose a two-phase regression testing methodology. In phase 1, we explore control flow and data flow analysis issues of database applications. Then, we propose an impact analysis technique that is based on dependencies that exist among the components of database applications. This analysis leads to selecting test cases from the initial test suite for regression testing the modified application. In phase 2, further reduction in the regression test cases is performed by using reduction algorithms. We present two such algorithms. The Graph Walk algorithm walks through the control flow graph of database modules and selects a safe set of test cases to retest. The Call Graph Firewall algorithm uses a firewall for the inter-procedural level. Finally, a maintenance environment for database applications is described. Our experience with this regression testing methodology shows that the impact analysis technique is adequate for selecting regression tests and that phase 2 techniques can be used for further reduction in the number of theses tests.

INTRODUCTION

Software maintenance involves changing programs as a result of errors, or alterations in user requirements. Regression testing is an important activity of software maintenance, which ensures that the modified software still satisfies its intended requirements as reported by Hartmann and Robson (1989). It attempts to revalidate modified software and ensure that new errors are not introduced into the previously tested code. Regression testing involves essentially four issues: change impact identification, test suite maintenance, test strategy, and test case selection. In this work, we concentrate on impact analysis and test selection for SQL-based systems. Regression testing is necessary for assuring the quality of a system after modifying it. Therefore, this activity must be performed by software maintainers. Ad hoc regression testing involves either rerunning all the test cases that are included in the test suite determined during the initial development of software (Select-All approach) or selecting a random subset of this initial test suite (Select-Random approach). But, the Select-Random approach is unreliable, since it might miss selecting test cases that reveal adverse effects of modifications. Hence, the Select-Random approach might compromise the quality of the modified system. On the other hand, the Select-All approach is expensive in terms of time and cost, since it usually includes many test cases that do not reveal the impact of the modification made to the system. Therefore, it is important to use regression testing methods that reduce the number of selected test cases (to save time and money), especially for large software systems, while maintaining the quality of the system. For example, Wong, Horgan, and Agrawal (1997) have reported useful results of a practical study of regression testing effectiveness using a software tool (ATAC).

Regression testing algorithms and approaches have been proposed for procedural and object-oriented programs. Examples of these algorithms and approaches are: incremental slicing algorithm proposed by Agrawal, Horgan, and Krauser (1993); slicing algorithms based on data flow testing and incremental data flow analysis described by Gupta, Harrold, and Soffa (1996) and Harrold and Soffa (1988); firewall-based approaches presented by Hsia et al. (1997), Kung et al. (1995), Leung and White (1990a and 1990b), and Lenung and White

(1992); stochastic search algorithms presented by Mansour and El-Fakih (1999); safe algorithm based on module dependence graph proposed by Rothermel and Harrold (1997) and Rothermel and Harold (1998); semantic differencing approach proposed by Binkley (1997); and textual differencing approach proposed by Vokolos and Frankl (1998). However, to the best of our knowledge, database programs have not been specifically dealt with in regression testing research. SQL-based database programs support a number of features that do not exactly apply in the cases of procedural and object-oriented programs. Examples of these features are: SQL statements, table constraints, exception programming, and table triggers. These features introduce new difficulties that hinder regression test selection.

SQL, the standard query language, stands as the heart of database applications modules as reported in ISO/IEC 9075 (1992). The usage of SQL in a procedural context has its implications. We categorize these implications into three categories: control dependencies, data flow dependencies, and component dependencies. The nature of SQL and the existence of table constraints lead to using exception handling techniques in database modules. Exception programming complicates control flow dependencies between statements in database modules. This complexity should be handled before applying any control flow based regression testing technique. Moreover, table triggers firing because of modifying SQL statements create implicit inter-modular control flow dependencies between modules. These dependencies should be explored before performing any inter-module regression testing.

The manipulation of database tables by different modules, using SQL, leads to a state-based behavior of modules. It also creates data flow dependencies between the modules. The dynamic behavior of SQL, in which the exact table rows manipulated is not known until run-time, makes it very difficult to trace such data dependencies. Furthermore, SQL manipulates database components such as tables and views. These facts create component dependencies between the various components handled by SQL statements and the modules in which the statements are located. These component dependency relations are transitive. Whenever a change is made to one component, this transitivity introduces a ripple effect of change.

In this paper, we propose a new two-phase methodology for regression testing SQL-based database applications. Phase 1 involves detecting modifications and performing change impact analysis. The impact analysis technique localizes the effects of change, identifies all the affected components, and selects a preliminary set of test cases that traverse modified components. Phase 2 involves running a test case reduction algorithm to further reduce the regression test cases selected in phase 1. In this work, we present two such algorithms. The first algorithm, Graph Walk, is a control flow based regression testing technique that utilizes control flow information, component dependencies, and impact analysis results. The second algorithm, Call Graph Firewall, utilizes data flow dependencies and is an adaptation of firewall-based regression testing techniques at the inter-procedural level. Furthermore, we develop a prototype maintenance tool and use it to empirically validate our proposed methodology.

The remainder of this paper is organized as follows: The next section includes a discussion of the structure of database applications and control flow issues of database modules. We then address the data-flow dependencies due to the manipulation of data stored in database tables followed by phase 1 of our regression testing methodology.Next, we present phase 2 and describe two alternative algorithms for the reduction of regression test cases selected in phase 1. In the next section, we present a maintenance tool for database applications that implement our regression testing methodology and empirically investigate the applicability of the methodology using the tool. Finally the conclusions are presented.

CONTROL FLOW MODELING

Background

Database systems have been accepted as a vital part of the information system infrastructure. They can be considered a mature technology whose characteristics have been covered in past manifestos. Although there are different variations of database systems implementation, we will limit our scope to the relational database systems because relational database systems are widely spread and the relational concepts are standardized.

SQL remains the most accepted and implemented interface language for relational database systems. SQL is designed to be a comprehensive language that includes statements for data definition, queries, updates, and view definition.

Lately, extensions to the SQL language were introduced. These extensions allow client (application program) requests to the server to perform lengthy, complex operations, with only the final results returned to the client. These SQL extensions were in the form of stored procedures and procedural language constructs that allowed significant application logic to be stored and executed on the server instead of on the client. Persistent Stored Modules (PSM) were published as an international standard in the form of a new part to the SQL-92 standard. This standard – ISO/IEC 9075-4, which appeared in 1995, was an extension to standard SQL for procedural language constructs, based on the best language concepts.

Control flow analysis of PSM code is different from that of conventional programming languages. Building control flow graphs for database modules differs slightly from building control flow graphs for conventional software. This difference results from the extensive usage of exceptions and condition handlers and the nature of the SQL language that is a key feature of database modules. Therefore, we should devise new modeling techniques to model the control transfers that are available in database modules.

The semantic of all SQL statements make them behave