Regression Testing of Database Applications[1]

Bassel Daou, Ramzi A. Haraty, Nash'at Mansour
Lebanese American University
P.O. Box 13-5053
Beirut, Lebanon
Email: rharaty, nmansour@lau.edu.lb

Keywords: Regression testing, database applications, and impact analysis.

Abstract

Database applications features such as SQL, exception programming, integrity constraints, and table triggers pose some 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. Finally, a maintenance environment for database applications is described. Our experience with the environment prototype shows promising results.

1 Introduction

Regression testing is an important activity of software maintenance, which ensures that the modified software still satisfies its intended requirements [10]. It is an expensive testing process that attempts to revalidate modified software and ensure that new errors are not introduced into previously tested code. Software revalidation involves essentially four issues: change impact identification, test suite maintenance, test strategy, and test case selection [7]. In database applications a number of new features is supported such as: 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. The usage of SQL in a procedural context has its implications and requirements. 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 imply the usage of exception handling techniques in database modules. Exception programming complicates control flow dependencies between statements in database modules. Moreover, table triggers firing because of modifying SQL statements create implicit inter-modular control flow dependencies between modules.

The manipulation of the database tables using SQL by different modules leads to a state-based behavior of modules. It also creates data flow dependencies between the modules

SQL manipulates database components such as tables and views. This fact creates component dependencies between the various components handled by SQL statements and the module in which the statement is located.

Regression testing algorithms and approaches have been proposed for procedural and object-oriented programs. Examples of these algorithms and approaches are: firewall concept presented in [11, 12, 13], incremental slicing algorithm proposed in [1], slicing algorithms based on data flow testing and incremental data flow analysis described in [4, 5], class firewall for regression testing object-oriented software presented in [7], safe algorithm based on module dependence graph proposed in [15, 16], semantic differencing approach proposed in [3], and textual differencing approach proposed in. However, to the best of our knowledge, database programs have not been specifically dealt with in regression testing research.

In this paper, we propose a new approach to regression testing of database applications. This approach is a 2-phase approach. 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 is a control flow based regression testing technique that utilizes control flow information, component dependencies, and impact analysis results. The second algorithm is an adaptation of the firewall regression testing technique on the inter-procedural level that utilizes data flow dependencies.

The remainder of this paper is organized as follows. The next section includes a discussion on the structure of database applications and control flow issues of database modules. Section 3 addresses the data-flow dependencies due to the manipulation of data stored in database tables. In section 4, we

[1] Proceedings of the ACM SAC 2001 Conference, Las Vegas, USA. March 11-14, 2000.

present phase 1 of our regression testing methodology. In section 5, we present phase 2 in which we give two alternative algorithms for the reduction of regression test cases selected in phase 1. In section 6, we present a maintenance tool for database applications that implement our regression testing methodology. In section 7, we empirically investigate the applicability of the methodology using the tool.

2 Database Applications

Database systems have been accepted as a vital part of the information system infrastructure. 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. Lately extensions to the SQL language were introduced. These SQL extensions were in the form of stored procedures and procedural language constructs that allowed significant application logic to be stored and executed in server instead of in the client. Persistent Stored Modules was published as an international Standard in the form of a new part to SQL-92 standard [9].

2.1 Control Flow Issues

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 like micro-transactions in that either they execute successfully, or they have no effects at all on the stored data [8].

A database module consists of one compound statement in which other compound statements are nested. Each compound statement has its exception handler. During execution, if an exception is raised from an SQL statement then the control is transferred from the current statement to the exception handler according to the type of the exception raised.

2.2 Suggested Technique

Each statement should be represented by a node in the control flow graph. These statements are either SQL statements or control statements or others. A compound statement contains a list of statements with one exception handler for all of these statements. Each of these statements is represented by a node. The compound statement contains two end statements one for successful endings and the other for unhandled exception results. If exception handling is not available, then all the exception links of these nodes will be linked to the unhandled exception end node.

If exception handling is available then the exception handler is modeled by a primary handler switch node to which all the exception links of the compound statement nodes are linked. Each specific exception handler is modeled by a predicate node that checks for the type of the exception. The exception predicate has two links: the first one is to the start node of the exception handler block and the second to the next handled exception.

3 Data Flow Analysis

Data flow analysis focuses on the occurrences of variables within the program. Each variable occurrence is classified as either definition occurrence or as use occurrence [14].

3.1 Data Flow Issues

The database plays an important role in holding the state of computation in database modules. The data generated by a statement is used by other statements in the same module or other modules; thus creating data flow relations. The main source of data in a relational database is tables.

To define the data flow relations created from the database usage we should decide on a level of granularity of the database variables in which we can trace their definition and their later use.

3.2. Suggestion and Solutions

One choice of the level of granularity is the column level. Since the number of columns is fixed and columns are used in SQL statements using their unique names, we can determine the column usage statically. A drawback of this choice is the fact that it does not discriminate between the usage of one particular column value belonging to some row and the usage of the same column but of a different row.

SQL statements use columns directly and indirectly or, in other words, explicitly and implicitly. These usages are either definition or retrieval. A table participating in master detail relations has a group of its columns referencing the primary key columns of the master table. Whenever these columns are defined the database implicitly checks that the master table contains a record that has its primary key column values matching the foreign key column values of the newly added record. So, whenever a new record is created the primary key columns of the master table are used.

We differentiate between five main usages of database columns. They are delete, insert, reference, select, and update. Reference and select usages are computational usages denoted by c-use. Update, delete and insert usages are define uses denoted d-use.

4 Impact Analysis

Software impact analysis estimates what will be affected in software or related documentation if a proposed software change is made [2].

In this section, we present phase 1 of our regression testing methodology that includes modification detection and impact analysis. In this phase we localize the effects of change, identify all affected components, and select a preliminary set of test cases that traverse modified components.

4.1 Change Identification

Change identification is the first step in change impact analysis. We differentiate between two types of changes in the database applications environment:

1- **Code Change**: This involves changes that can be made to the code of the database modules
2- **Database Component Change**: This change involves the changes that could be made to the definition of the database components in general.

4.2 Change Impact Identification

A change made to one component affects other database components due to component dependencies. Therefore, to identify the impact of change, we should identify the dependencies that exist between database application components and then find the wave effect of change due to the transitivity of the dependency relations. The Component Firewall technique presented in this section is used to determine all the affected database components. .

A component firewall is a set of affected modules when some changes are made to any of the database components. A database component is marked as modified and is included in the component firewall if one of the following conditions is satisfied:

1- Its definition is modified.
2- It is deleted.
3- It is dependent on a modified or deleted component.
4- It became dependent on new or modified components in the new system such as triggers and constraints.

All database components selected by the Component Firewall Algorithm are marked as affected components. Affected module components are classified alone so that we can select a test case passing through them to become a part of the results acquired in phase 1 of our regression testing methodology.

In Figure 1, we sketch an outline of the component firewall building algorithm. This algorithm takes the old and new schemas and returns a list of components that constructs the component firewall.

```
Component_Firewall(old_schema, new_schema)

    Denote by L the list of components in the firewall
    Denote by ML the list of modified and deleted components
    Denote by NL the list of new components

    Compare(old_shema, new_schema, ML, NL)
    For each modified component C in ML
        Add C to L
        For each component X dependent on C in new_schema
                If X belongs to Old_shema then
                        Add X to L
    For each new component C in NL
        For each dependent component X on C in new_schema
                If X belongs to Old_shema then
                        Add X to L
    L := Transitive_Closure(L, Old_schema)
    Return L
```

Figure 1 The Component Firewall Algorithm.

Module *Compare* is responsible for performing change identification. It takes the old and new database schemas and returns two lists of components: one for the modified and deleted components and the other for the newly added ones. Module *Transitive_closure* takes a list of components and the database schema and returns the transitive closure of the dependent components.

5 Test Case Reduction

The test cases passing the modules that are included in the firewall are selected for regression testing. However, this will result in a large number of test cases. The component firewall does not give us hints to discriminate between the test cases passing through a module included in the firewall.

Therefore, we have to think of new techniques to reduce the number of test cases selected in phase 1. In this section, we will discuss two such techniques. We call the first technique a Graph Walk technique. The second technique is call graph firewall. It is an adaptation of the firewall regression testing technique proposed by Leung and White [11, 12].

5.1 Graph Walk Technique

In this technique, we use control flow graphs of all modules in the application and its modified version, and trace-information linked to control flow nodes. We also utilize the dependency created between statements and various database components

To perform the technique on a certain module the technique traverses the control flow of the module and its modified version. When a pair of nodes N and N* in the graphs of the original module and its modified version are respectively discovered, such that the statements associated with N and N* are different, the technique selects all tests from the test suite that reached N in the original program. For two node N and N* to be different, at least one of these conditions should be satisfied.

1- N and N* are lexically different,
2- N uses a modified component,

3- N uses a component that is not used by N*, or
4- N* uses a component that is not used by N.

To extend the technique to the inter-module level, we should change condition two to become: N uses a modified non-module component. Moreover, for each module call linked to a control flow graph node N we should perform the graph walk algorithm recursively on this module and intersect the result with the test cases passing through node N.

5.2 Call Graph Firewall

Leung and White [11] present a selective retest technique aimed specifically at inter-procedural regression testing that deals with both code and specification changes. Their technique determines where to place a firewall around modified code modules. Where test selection from regression test suite is concerned, the technique selects unit tests for modified modules that lie within the firewall, and integration tests for groups of interfacing modules that lie within the firewall. Leung and White [11] extend their technique to handle interactions involving global variables.

Implementing the firewall concepts for database applications has three requirements:

1- Database application call graph.
2- Data flow dependencies between interfacing modules resulting from database tables usages.
3- List of modified database modules.

The call graph links a database module to all the modules that it calls. It should include links to table triggers modules in case the module contains statements that causes these triggers to execute.

6 Support System

We have implemented a database applications maintenance tool as a support system for this research. The objective of our support system is to prove the applicability of the concepts presented. The developed system helps database application maintainers understand these applications, identify code changes, support software updates, enhance, and detect change effects. It mainly helps create a test environment and select regression test cases to be rerun when a change is made to the application using our 2-phase regression testing methodology.

The system is made for ORACLE database applications programmed using PL/SQL language. Our maintenance tool is composed of five parts: module analysis, database analysis, test environment setup, impact analysis and regression test selection, and test case reduction.

7 Empirical Results

To empirically investigate the use of our regression testing methodology, we have performed a study on a prototype of a payroll database application

7.1 Experimental Design

We use a prototype of payroll database application with a number of test cases used to test its various modules and constructs. We propose a random number of modifications to the application. Then, we study each modification alone using our maintenance tool and report the affected modules and the test cases that should be rerun according to the regression testing techniques implemented in the tool. The test suite used to test this application contains fifty test cases selected using a specification based test adequacy criterion.

7.2 Summary of Results

In Table 1, we present a summary of the cases presented in section 7.3. We classify these results into two parts. In the first part, we give the results of phase one of our regression testing methodology. In the second part, we give the results of the phase two.

	Phase 1			**Phase 2**	
				Selected Tests	
Modification Cases	**Directly Affected Modules**	**Indirectly Affected Modules**	**Selected Tests**	**Graph Walk**	**Call Graph Firewall**
1- Delete trigger	5	2	18	18	11
2- Modify function	2	2	14	14	13
3- Drop constraint	1	0	4	4	4
4- Drop constraint	1	0	4	4	4
5- Drop constraint	1	2	14	14	7
6- Add constraint	1	2	14	14	7
7- Add trigger	3	0	4	4	4
8- Change column type	1	4	14	12	12
9- Modify function code	1	3	14	5	5
10- Modify function code	1	3	14	2	5

Phase 1 results include a count of the following:
1- Directly affected modules.
2- Indirectly affected modules.
3- Test cases traversing affected modules.

Phase 2 results include a count of the following:
1- Test cases selected by the GraphWalk technique.
2- Test cases selected by the Call Graph Firewall technique.

Table 1 Summary of Results. (Total test cases = 50)

7.3 Discussion of Results

In phase 1 of our regression testing methodology, impact analysis does a good deal of the test selection job. It selects the test cases that traverse affected modules. Out of the 50 test cases used to test the application we have 18 test cases selected at most, which is a 36% ratio (refer to Table 1). The best case is 4 test cases out of 50, which is 8%. On average 11.4 test cases are selected, which is 22.4%. This ratio depends on the number of affected modules per modification and the distribution of test cases within the modules. The number of affected modules per modification depends on the level of interaction between the modules and the various database components. On the other hand, the distribution of test cases within the modules depends on the testing criteria used to initially test the application.

In phase 2, we have two alternative techniques to reduce the number of regression test cases selected in phase 1. The GraphWalk algorithm works on the statement level and can be extended to the procedural level. In seven modification cases, the reduction has not been effective (refer to table 7.1). However, in the remaining three modification cases the reduction has been more evident. We account this behavior to modification types and to modules structure. With non-module component modifications, the number of affected modules and statements is usually high. This results in the selection of a high number of regression test cases. On the other hand, when few statements are affected within the control flow branches of a module, the GraphWalk selects the test cases traversing these branches only and eliminates the other test cases. Therefore, with code modification like the case of modifications 9 and 10 the reduction of test cases will be more evident specially if the affected code lies deep in the branching structure of the module.

The second alternative technique for reduction of regression test cases in the second phase of our regression test methodology is the Call Graph Firewall technique. The test cases selected by the firewall regression testing technique are composed of two types of tests: unit tests and integration tests. Unit tests are tests used to test only the directly affected modules. Integration tests are test cases passing to the directly affected modules from higher modules in the call graph. These test cases are selected when there are data flow interactions between modules in the call graph. In six modification cases, the Call Graph Firewall algorithm has been able to reduce the number of selected test cases. In these cases, the ratio of the indirectly affected modules to those directly affected is relatively high. The number of indirectly affected modules is higher when the affected modules lie deep in the hierarchy of the call graph. With modular applications having a hierarchical structure there is more probability that the modified modules would lie within the hierarchy. Therefore, the Call Graph Firewall technique is effective with modular applications.

8 Conclusion and Further Work

We presented a two-phase regression testing methodology for database applications. In phase 1, we suggested a technique for modification detection and modification impact analysis in which we determined affected modules and test cases traversing them. In phase 2, we presented two alternative techniques for the reduction of the regression test cases selected in phase 1. In the first technique, we presented a statement based regression testing algorithm, the graph walk algorithm, that extends to the inter-procedural level. In the second technique, we adapted the firewall algorithm to database applications. In addition, we developed a support system and used it for the experimental work.

References

[1] Agrawal, H., Horgan, J.R., Krauser, E.W., 1993. Incremental Regression Testing. *Proceeding of International Conference on Software Maintenance*, 348-357.

[2] Arnold, R. S., and Bohner, S. A. 1996. *Software Change Impact Analysis*. IEEE press.

[3] Binkley, D., 1997. Semantics Guided Regression Test Cost Reduction. *IEEE Transactions on Software Engineering*, 23(8), 498-516.

[4] Gupta, R., Harrold, M.J., Soffa, M.L., 1996. Program Slicing-Based Regression Testing Techniques. *Software Testing, Verification and Reliability*, 6(2), 83-111.

[5] Harrold, M.J., Soffa, M.L., 1988. An Incremental Approach to Unit Testing during Maintenance. *Proceeding of International Conference on Software Maintenance*, 362-367.

[6] Hartmann, J., and Robson, D.J. 1989. Revalidation during the Software Maintenance Phase. *Proceeding of International Conference on Software Maintenance*, 70-79.

[7] Hsia P., Li X., Kung D.C., Hsu C-T, Li L., Toyoshima Y., and Chen C. 1997. A Technique for the Selective Revalidation of OO Software. *Software Maintenance: Research and Practice*, Vol. 9, 1997, 217-233.

[8] ISO/IEC 9075: 1992. Information Technology – Database Languages – SQL.

[9] ISO/IEC 9075-4: 1995. Information Technology – Database Language – SQL Part 4: Persistent Stored Modules (SQL/PSM).

[10] Kung D.C., Gao J., Hsia P., Wen F., Toyoshima Y., and Chen C. 1995.Class firewall, test order, and regression testing of object-oriented programs. *Journal of. Object-oriented Programming*, 8(2), pp. 51-56.

[11] Leung, H.K.N., and White, L. 1990a. A Study of Integration Testing and Software Regression at the Integration Level. *Proceeding of International Conference on Software Maintenance,* 290-300.

[12] Leung, H.K.N., and White, L. 1990b. Insights into Testing and Regression Testing Global Variables. *Software Maintenance: Research and Experience*, Vol. 2, 209-221.

[13] Leung, H.K.N., White, L., 1992. A Firewall Concept for both Control-Flow and Data-Flow in Regression Integration Testing. Proceeding of International *Conference on Software Maintenance,* 262-271.

[14] Rapps S., and Weyuker E. J. 1985. Selecting Software Test Data Using Data Flow Information. *IEEE Transactions on Software Engineering,* 24(6), June, 401-419.

[15] Rothermel, G., Harrold, M.J., 1997. A Safe, Efficient Regression Test Selection Technique. *ACM Transactions on Software Engineering and Methodology*, 6(2), 173-210.