Efficient Regression Test Model for Object Oriented Software

Swarna Lata Pati
College of Engg. & Tech, Bhubaneswar

Abstract: This paper presents an efficient regression testing model with an integration testing strategy for testing of object oriented software. To make the regression test more effective and efficient test cases are generated using a tool JUnit. JUnit framework is used to test the original version of the software called regression test selection and this model selects every test case or test suite that may reveal a fault or bug or defect in the software. This model serves as a basis for ordering of classes, from which prioritization test cases are derived. The prioritization of test cases is done depending upon fault exposing capabilities. A set of empirical studies that demonstrate that the presented model can be effective and efficient in reducing the size cost resource and time of regression test.

Keywords: Regression testing, JUNIT test case generation, fault, software maintenance, efficient.

I. INTRODUCTION

Regression testing is applied to modified software to provide confidence that the changed parts behave as intended and the unchanged parts have not been affected much by any modification. Regression testing is becoming one of the key aspects of object oriented methodologies to full fill the need of the hour testable and trustable object oriented systems. Design for testability is a necessary basis for final product reliability. Design for testability aims at integrating design and testing in the same process. It includes the problem of Test planning from early design stage test, as the frame work chosen is based on test driven development [12]. This model presents a strategy for test plan, types of dependencies, strategy for unit test, as well as integration strategy. Types of dependencies among classes, methods help ordering of classes for integration process. It helps in minimization of test cases. The presented model also helps refactoring of testing resources during system integration for minimization of duration of integration. We need a test plan to test the testable software. The introduction of test plan benefits us to determine the effort requires to validate the quality of UUT (unit under test). In addition test estimation, test scope, test strategy (JUnit[5]) is determined in a test plan. It can be reused in future to achieve cost and time effectiveness.

This paper is organised as follows: section II discusses the overview of the regression test model; section III discusses the structural test dependency which is used to establish relationship between class and method, which serves as a basis for defining unit testing. Unit testing is also called component testing, the process of testing individual components in the system. Section IV discusses JUNIT testing framework and prioritization, Section V discusses Integration and regression strategy section VI discusses empirical studies section VII discusses the conclusion and future work.

II. REGRESSION TEST MODEL

Regression test model (RTM) comprises of units as depicted in the diagram-1. It contains a test plan which is introduced at an early stage of design. This test plan helps to assess the test strategy, define testing effort and cost, and define the test criteria, plan resource and environment for regression test. The next phase of the model is to design test cases. The test cases are designed from structural dependency. Next to test generation using automated unit(s) test using JUNIT [11] framework as I propose to conduct my studies in Java software. During that test data are to be prepared. Thereafter RUN own automated unit(s) test with test data. Implement code to pass test(s) or refactor as needed, compare the results with test cases and produce test report. Regression testing is applied to modified software. To make the regression test more effective and efficient test cases are generated using a automated tool JUnit framework used to test the original version of the software called regression test selection and this model selects every test case in test suite that may reveal a fault/bug/defect in the modified software[1]. The test suite which reveals fault earlier is to get most priority. This model ensure that reuse of test cases useful in a maintenance context as well as for regression testing.

This paper presents a flexible regression test model for regression testing. The flexibility is ensured by inputting to JUnit from structural test dependencies as well as form test case prioritization depending upon the circumstances. Regression testing is an expensive process model and it can be made cost effective with the help of automated support. In my propose model I have used JUNIT[6,10] as automated testing tool, tests can be automatically rerun. It is an acceptable unit-testing framework helps to generate whole test suites quickly that validate the program with the click of button. The
complete test set is executed whenever new code is not accepted until all test run successfully.

REGRESSION TEST PROCESS MODEL

![Diagram of Regression Test Process Model]

III STRUCTURAL TEST DEPENDENCY

A testing type is a standard test procedure that gives an expected test outcome. In this model I have proposed unit testing followed by integration testing and system testing. The unit test is the smallest piece of visible software. In this model, top-down approach of unit testing is used. The test exit is selected after the completion of successful test phase. Run rate and pass rate for each test is calculated by using simple ratio formula i.e. run rate = no. of test case executed/total no of test cases and Pass rate = no. of test cases/test cases executed. Run rate is to be 100% and pass rate is to be high it depends upon the scope of the project. In this model Top down approach of unit testing is used. In this approach the system is broken down into component and sub component. To model test dependencies among the modules of code the concepts of component, integration, method, stubs are introduced, these are used in rest of the paper[14].

Component: It is a basic test unit. It corresponds to a class or specific method of a class in a refined design.

Integration testing: It is the way in which test is conducted to integrate components into the system. The integration always uses incremental steps. The main objective to achieve cost-effective integration of components is the minimization of number of stubs to be written.

Stub: a dummy component used to simulate the behavior of a real world component. The test of a unit u1 that calls a unit u2, that is not tested, replacement of u2 by a dummy component called stub. A specific stub is written if it simulates u2’s behavior relative to u1 use.

The testing effort depends upon the number of stubs that have to be written in integration strategy. The number of stubs is calculated depends upon types of stub i.e. realistic and specific. If realistic stubs are written we need one stub instead of two to test two number of components.

Regression testing: this is used when components of the system evolve or when new components and functionalities are added to the system. It tries to assert that changes are correct and no regression bugs appear in the system due to recent addition. Generally previous test cases are used to guarantee that the system has not regression in terms of testing quality.

Test case design from test dependency

For system S, classes and methods from the system design are related if there some common objects. For example for two components c1 and c2, c1 related c2 if c1 uses some objects from c2 or inherits from c2. So we cannot really test c1 unless c2 is tested. So test cases are generated for c2. Test cases can cover private or protected parts of the system where the tests are lined to private implementation Public interfaces parts of the system where test concerns the public aspect of the class. The test cases related to private class are less reusable in comparison to test cases related to public class. Test cases are designed depending upon types of dependency. Depending upon the level of detail reached by design one may define test dependency between
classes, methods and between class and method. We may go for three types of test dependency as follows:

Class-to-Class: It is the first test dependency that can be induced from a design.

Method-to-class: A method –to-class from m to A test dependency is induced if a method m has an object of a class A declared in its signature. Polymorphism is modeled by having m transitively dependent on A subclass through the inheritance dependency link.

Method-to-Method: This type of test dependency can inferred only if detail exist on the implementation body of a method. A method-to-method dependency from m1 of class A to m2 method of class B exists if m1 applies the method m2 to an object of class B.

The diagrammatic representation to find out test dependency from UML diagram and associated relationships are depicted in diagram 2, 3, 4, 5 respectively at Appendix-1.

IV. JUNIT TEST CASES

Using JUnit tool incrementally one can build a test suite and enhance development efforts. It defines how to structure your test cases and provides the test tool to run them. Each test runs in its own fixture and Junit[5] calls setUp and tearDown for test so that the side effects among test run can be avoided. JUnit supports two ways of running single test: static and dynamic. To run more than one test case is to run a test suite the suite method is the main method inside that one can add tests to be run to a test suite object and return it. Test suite and test case both implement an interface which defines the method to run a test JUnit provides GUI to run test suite. We have designed JUnit test suites that invoke sequences of test classes to avoid initial startup cost which is attributable to achieve cost effectiveness. We have taken the test cases at coarse-granularity[9] at class test level method and fine-granularity level of JUnit methods within test class classes as test –method level. During development process when you need to add new functionalities to the system write the test first then develop similarly during debugging to eliminate defect write a test , if it succeed the code is working , debug until the test succeeds. So the test methodology is Test Driven Test (TDD). Since JUnit is an open source Unit testing tool for Java program achievement of effectiveness of the test matters. Test case prioritization can be effective by revealing the fault detection early our experiment shows. We have therefore designed and conducted a controlled experiment examining whether prioritization of test case can be effective in Java software. JUnit test involves Java classes that contain one or more test methods and that are grouped into test suites. Test method is a minimal unit of test code. The components of JUnit framework play vital role to conduct experiment. The components are – Test suite, Test fixture, test runner, test result formatter and assertions. Section VI discusses the experimental result.

V TEST INTEGRATION STRATEGY

The integration strategy is based on the decomposition of dependency diagram [13]. As result components are ordered with respect to the minimization of stubs. The diagram is decomposed according to strongly connected component and organizes the test by minimizing the stubs to be written. The process of integration involves building a system from its component and testing the resultant system for problems that arise from component interactions. The components that are integrated may be reusable or newly developed components. At the beginning the skeleton of the system is prepared then components are added one by one using top down approach. Let c1, c2, c3, c4 are components and t1, t2, t3, t4, t5 are set of test sequence. Let at the beginning three tests t1, t2, t3 are first run on a system composed of component c1 and c2 .If finds bug , these are corrected and another component c3 is added and test t4 is also run on the system and finally component c4 is added and tested using existing and new test t5 and so on. However prioritization of component depends upon average fault detection capabilities.

VI. EMPIRICAL STUDIES EVALUATION

To evaluate our approach we used proposed regression test model presented in this paper to perform empirical studies. Five small subjects are taken into considerations written by author. And its modified version also. The study utilized the code for two successive versions. These are chatclient.java (login procedure in a c-s application), calc.java (to perform basic arithmetic operations), libsys.java (library program) and studexm.java (student result), empdtl.java (employee details). JUnit regression testing framework is used to implement unit testing in java and accelerates programming speed and increase the quality of code. JUnit version 4 is integrated with Eclipse as it supports annotation[7]. The empirical procedure used as follows:

Step 1- a fixed set of objects used as base line for running tests. Program be instrumented to support the technique needed. We instrumented class file, all blocks and methods using byte code analyzer.

Step2-bundle a few unit test cases and prepare it to run with @Run @ Suite annotations. Prior to that coverage information [2], fault exposing information and prioritization of test cases[1] are determined.

Step3 –JUnit classes are used in writing and testing Junit classes. Before that the change information must be there to find the difference in method and class with its previous version.

Step4- As described test cases were obtained from each objects from distribution and we consider test suite at the two level of granularity[2,3]. To execute and validate test cases distribution we created test scripts and invoke JUnit test cases at test-class and test-method level[4]. Record all output from test execution and compare outputs with those for the previous version.
Step4-TestRunner is used for executing test cases.

Table 1 and 2 provides information about the experimental output in terms of coverage information which are given hereafter in the appendix along with graphical presentation.(vide appendix-A)

VII. CONCLUSION AND FUTURE WORK

I have presented a regression test model to conduct regression test using JUnit as test tool. The model is used for Java software. A blending of prioritization technique and test dependency from UML diagram is applied to JUnit test suites. Our efficiency of model confirm several previous findings but some difference due to steps followed to conduct experiment. These differences may be program specific or JUnit testing model. I hope research scholars may use Junit testing tool for regression test and can improve the effectiveness and efficiency of the regression testing process. This involves a return on investment analysis to determent cost-benefit tradeoff between the increase in development time and resulting improvement in software quality.

REFERENCES


GRAPHICAL REPRESENTATION OF TABLE DATA

APPENDIX-1

Class A

Method mA1 in Class A

Method node in class node

Method _to_Class

Method _to_method

DIAGRAM-2

DIAGRAM-3