International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

A Study for Regression Testing Techniques and Tools

[1]Passant Kandil, [2]Sherin Moussa, [3]Nagwa Badr

Department of Information Systems,
Ain Shams University, Cairo, Egypt

Email: [1]: passant.kandil@cis.asu.edu.eg, [2]sherinmoussa@cis.asu.edu.eg,
[3]nagwa.badr@cis.asu.edu.eg

Abstract. Regression testing is a part of the software testing activity, which is an important activity of the software development life cycle and the maintenance process. It is carried out to ensure that changes made in the fixes or any enhancement changes are not influencing the previously working functionality. Regression testing is mostly done by re-running existing test cases against the modified code to determine whether the changes affect anything. This requires a lot of cost and time, which increases as the size and the complexity of the software increases. Instead of re-running all the test cases, a number of different approaches were studied to solve regression-testing problems. There has been an explosion in the use of data mining techniques in the exploration and analysis of large quantities of data in order to discover meaningful patterns and rules. Data mining models were introduced for software testing to design a minimal set of regression tests. This helps solving regression testing problems with large-scale systems that are usually accompanied by thousands set of test cases, where it is considered impossible to re-run all of them each time a system update is applied. Therefore, data mining is investigated to handle such cases. In this paper, we investigate the different techniques proposed to solve the regression testing problems, where a comprehensive study is conducted for analysis and evaluation. We also discuss the tools presented in market for the regression testing. Finally, we present our proposed approach for regression testing using data mining techniques. The main advantage of this new approach is that it can be applied on large-scale systems having thousands of test cases. The proposed regression-testing algorithm considers time and cost constraints with no human intervention.

Keywords: *Software Testing, Regression Testing, Large-scale Systems, Test Cases Prioritization & Selection, and Data Mining.*

* Corresponding Author:
Passant Kandil,
Department of Information Systems, Ain Shams University,
Cairo, Egypt,
Email: Passant.kandil@cis.asu.edu.eg Tel: (202) 6831231

1. Introduction

Software testing is a significant part of software engineering. It typically consumes 40 - 50% of development efforts, where this percentage increases for systems that need higher levels of reliability

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

[1]. The purpose of testing is operating the software under certain conditions to detect errors in order to fix the problems in the system and ensures that the system meets the requirements and does what it is expected to do in the intended environment. Different types of testing exist like Unit Testing, Integration testing… etc. Each occurs for a certain scope and for a certain specification.
Regression testing is one type of software testing that is performed with changes of an existing software. It is essential to provide confidence that such changes, which are newly introduced, do not obstruct the behavior of the unchanged existing parts of the software. Regression testing is costly, and represents a very important problem in the software development. Common methods of regression testing are re-running previously completed test cases and then checking whether the program behavior has been changed or the previously fixed faults have re-emerged. This requires a lot of cost and time as the size and complexity of the software increase. Different techniques were presented to solve regression-testing problems like regression test cases selection, regression test cases reduction and regression test cases prioritization.

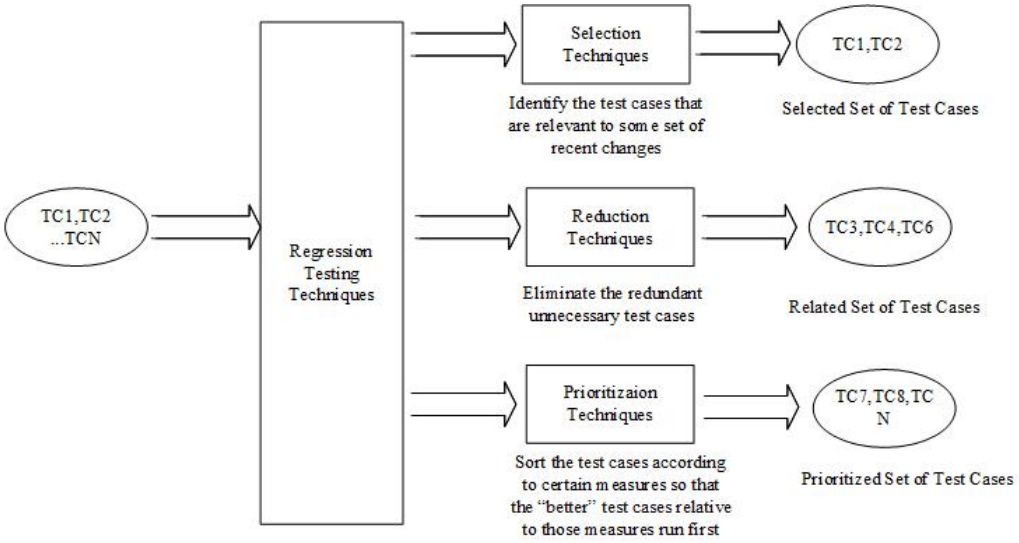

Figure 1: Regression Testing Approaches

The main concerns of these techniques when conducting regression tests are:

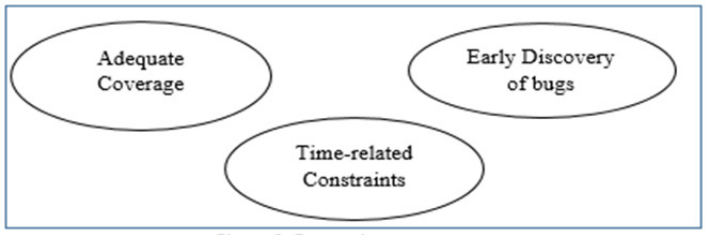

Figure 2 : Regression techniques Constraints

- Adequate coverage without wasting time.

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

- Software bugs and errors resulting from introducing the new changes are discovered earlier.
- Terminate regression testing if it is required at any instance for time-related constraints and alerted with the most expected software bugs and errors.

During the last several decades, new changes in software engineering have already become ordinary with the development of modern methods, technologies, and tools, which allow the functionality and complexity of modern systems, grow exponentially. Accordingly, as the functionality and complexity of the system grow, the cost and complexity of testing such a system grow as well. This limitation forces the consideration of techniques that seek to minimize the time, effort and cost required for regression testing in various ways, without sacrificing the thoroughness of regression testing. Another challenge is to handle the scalable data represented in the huge number of test cases associated with the large-scale systems that are currently developed due to the advancements in technologies. Simultaneously, there has been an explosion in the use of data mining techniques in the analysis of large quantities of data in order to discover meaningful patterns and rules. This allowed the potential use of the data mining algorithms to solve regression testing problems.
The rest of the paper is organized as follows: Section two introduces the parameters of evaluation used to study and evaluate the different proposed techniques. Section three investigates the different techniques used in regression test cases selection, reduction and prioritization, with a detailed analysis of the main issues and concerns fronting these techniques. Section four illustrates applying data mining techniques to regression testing. Section five presents the tools used in the market for regression testing. Sections six analyzes the different challenges of regression testing, emphasizing the current research gap in this field. Section seven, presents our proposed approach of regression testing for large-scale systems. Finally, we conclude our study.

2. Parameters of Evaluation of Regression Testing

Several parameters have been used throughout the recent research to evaluate the different regression testing techniques. Parameters have varied depending on whether the technique applies selection, reduction or prioritization.

2.1 Selection Evaluation Parameters:

Many studies have deducted some evaluation parameters of the selection techniques. Those parameters represent a set of basis in which selective techniques can be compared and evaluated.

1. **Inclusiveness:** This parameter measures the extent to which a selective re-test strategy S selects modification-revealing tests from the initial test suit T for inclusion in T′, where a test $Ti \in T$ is a modification revealing if it produces different outputs in P and P[2].

2. **Efficiency:** This parameter measures the efficiency of the selection algorithm in terms of space and time requirements. Space efficiency is affected by the test history and program analysis information. It varies the efficiency of S with the size of test cases that a method stores, as well as with the computational cost of that method [2].

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

3. **Generality:** This parameter measures the ability of a selective re-test strategy to function in a wide and practical range of situations[2].

4. **Accountability:** This parameter measures the extent to which a selective re-test strategy promotes the use of structural coverage criteria, as it increases the effectiveness of testing [2].

5. **Precision:** it represents the accuracy degree of selection, which measures the extent to which a selective re-test strategy S ignores test cases that are non-modification-revealing[2], [3].

$$Percision = \frac{|T'F|}{|T'|} \quad \ldots (1)$$

Where T'F is the set of failed test cases from T', which is the set of selected test cases.

6. **Recall:** it represents the completeness of test selection, which measures the proportion of selected failed tests in all failed tests [3].

$$Recall = \frac{|T'F|}{|T|} \ldots\ldots\ldots (2)$$

Where T'F is the set of failed test cases from T, which is the set of all failed test cases.

7. **F- Measure:** This parameter evaluates the integrative benefit of the precision and recall measures by the combination of the two parameters [3].

$$FMeasure = \frac{2 * Precision * Recall}{Precision + Recall} \ldots (3)$$

2.2 Reduction Evaluation Parameters:

Different evaluation parameters were emerged to evaluate the reduction techniques.

1. **Test Suite Size Reduction (TSSR):** It determines the percentage of the test suite reduction by using the following equation [4], [5],[6]:

$$TSSR = \frac{|TSorig| - |TSred|}{|TSorig|} * 100\% \ldots\ldots (4)$$

Where |TSorig|, |TSred| represents the sizes of the original and reduced test suite.

2. **Fault Detection Capability (FDC) Loss**: it determines the percentage of the test suite fault detection capability loss by using the following equation [4],[6]:

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

$$FDC = \frac{|Forig| - |Fred|}{|Forig|} * 100\% \text{..... (5)}$$

Where |Forig| and |Fred| represents the total number of unique fault revealed by the original and reduced test suites.

3. **Percentage of Test Suite Reduction**: It represents the percentage by which the test suite was reduced from the original suite [5],[6] :

$$100 * (1 - \frac{size\ reduced}{size\ original}) \text{... (6)}$$

4. **Fault Detection Rate**: it represents the percentage by which the rate of faults is detected [6],[5]:

$$100 * (\frac{Faults\ detected\ reduced}{Faults\ detetced\ original}) \text{..... (7)}$$

2.3 Prioritization Evaluation Parameters:

Another group of evaluation parameters were used to evaluate test cases prioritization techniques.

1. **APFD (Average Percentage Fault Detection):** It is a metric used to measure the test suite's fault detection rate. The fault detection rate is a measure of how quickly faults are detected during the testing process. The closer the value is to 100, the better the prioritization technique is [7].

$$APFD = 1 - \frac{(TF1 + TF2 + \text{.....} + TFm)}{nm} + 1/2n \text{..... (8)}$$

where: m -> is the number of faults contained in the program under test P, n -> is the total number of test cases, and TFi -> is the position of the first test in T that exposes fault i.

2. **APFDc (Average Percentage Fault Detection with cost):** It is a metric, which incorporates not just the rate of fault detection but also the severity of detected faults and the expense of executing test cases [7].

$$APFDc = \frac{\sum_{i=1}^{m}(fi*(\sum_{j=TFi}^{n} tj - \frac{1}{2}t_{TFi})}{\sum_{i=1}^{n} t_i * \sum_{i=1}^{m} f_i} \text{..... (8)}$$

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

Where T is the set of n test cases with costs t1, tn, and F is the set of m faults with severity values f1.., fm. For ordering T', let TFi be the order of the first test case that reveals the index fault.

3. **EVOMO (Evolution-aware economic cost model):** A cost model that captures the costs and benefits of regression testing methodologies and how much revenue they help organizations obtain. EVOMO involves two equations: first that captures costs related to the salaries of the engineers who perform regression testing (to translate time spent into monetary values). The second one that captures revenue gains or losses related to changes in system release time (to translate time-to-release into monetary values) [8].

   1. $Cost = PS * \sum_{i=2}^{n}(CS(i) + \mathrm{COi\,(i)} + \mathrm{COr\,(i)} + \mathrm{b(i)\_CVi\,(i)} + \mathrm{c(i)\_CF(i)})$….. (9)

   2. $Benefit = \mathrm{REV} * \sum_{i=2}^{n}(\mathrm{ED(i)} - (\mathrm{CS(i)} + \mathrm{COi\,(i)} + \mathrm{COr\,(i)} + \mathrm{ain} * \mathrm{CAin\,(i-1)} + \mathrm{atr\,(i-1)\_CAtr\,(i-1)} + \mathrm{CR(i)} + \mathrm{b(i)\_(CE(i)} + \mathrm{CVi\,(i)} + \mathrm{CVd\,(i))} + \mathrm{CD(i)}))$……(10)

   Where:
   - *PS is Average hourly programmer's salary in dollars per unit u.*
   - *CS (i) is time to perform setup activities required to test Si.*
   - *COi(i) is Time to identify tests that are obsolete for Si.*
   - *COr (i) is time to repair obsolete tests for Si.*
   - *b(i) is Coefficient to capture reductions in costs of executing and validating test cases.*
   - *CVi (i) is Human time for inspecting the results of test cases.*
   - *c(i) is Number of faults that are not detected by a test suite applied to Si.*
   - *CF(i) is Cost of missed faults after delivery of Si.*
   - *REV is Revenue in dollars per unit u.*
   - *ED(i) Expected time-to-delivery for Sw system when testing begins.*
   - *ain (i) Coefficient to capture reductions in costs of instrumentation for Si due to the use of incremental analysis techniques.*
   - *atr (i) Coefficient to capture reductions in costs of trace collection for Si due to the use of incremental analysis techniques.*
   - *CAin(i) is Time to instrument all units in i.*
   - *CAtr (i) is Time to collect traces for test cases in Si−1.*
   - *CR(i) is Time to execute a prioritization technique on Si.*
   - *CE(i) is Time to execute test cases on Si.*
   - *CVd (i) is Time to use tools to check outputs of test cases on Si.*
   - *CD(i) is Cost of delayed fault detection feedback on Si.*
   - *S is the SW system and I is Index denoting a release Si of S.*

4. **RFFT (Reduction Factor For Time):** It reflects how fast is the reduced test suite by measuring the difference of execution time between the original test suite and the reduced test suite. An RFFT of Zero means that T and Tr execute for the same length of time [9].

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

$$\text{RFFT (T, Tr)} = \text{time (T)} - \text{time (Tr)} / \text{time (T)} \quad \ldots (11)$$

Where T is a given test suite and Tr is its reduced form.

5. **RFFS (Reduction Factor For Size**): It reflects the percentage of original test cases that continue after reduction. An RFFS of Zero means that the algorithm removed none of the test cases; while an RFFS near 1 means that the reducer removed many test cases [9].

$$\text{RFFS (T, Tr)} = |\text{T}| - |\text{Tr}| / |\text{T}| \ldots (12)$$

Where T is a given test suite and Tr is its reduced form.

6. **CE (Coverage Effectiveness):** This measure determines the cumulative coverage of the tests over time; it takes the input of a test suite T and a time l and returns the total number of requirements covered by T after running for l time units [9].

7. **PTR (Problem Tracking Report):** it measures the effective prioritization technique by putting the test cases that are most likely has a fault equal to number of test cases detect a fault/Total number of test cases[10].

$$Ptr(t,p) = \frac{nd}{n} \ldots\ldots (13)$$

Where t is test suite under evaluation, nd is the total number of test cases needed to detect faults and n is the total number of test cases under test.

3. Regression Testing Techniques

In this section, we present the different techniques studied to solve the regression testing problems. Different techniques have been studied to solve regression testing problems such as Test Cases Reduction which permanently eliminates test cases from the test suite, Test cases prioritization which orders the test cases by certain measures and test cases selection which seeks to select the test cases that are relevant to some set of recent changes.

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

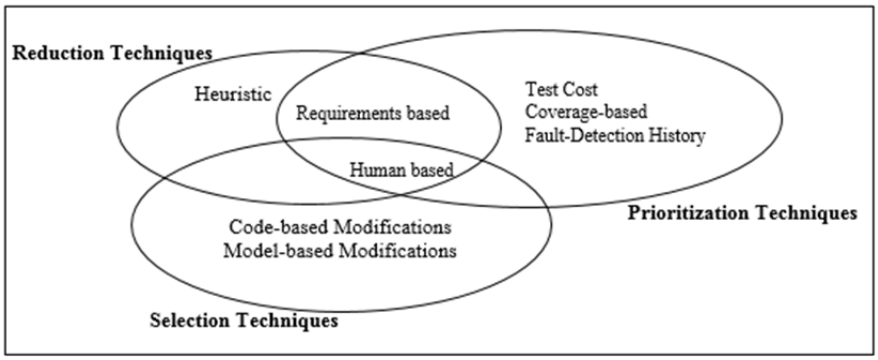

Figure 3: Different Regression Testing Technique

3.1 Test Cases Selection Techniques

Regression test cases selection techniques aim at selecting a relevant subset of test cases from the initial test suite instead of re-running the complete test suite, which minimizes both regression testing time and effort without sacrificing the thoroughness of regression testing. Different techniques were considered for test cases selection, where most of them apply one of the following approaches:

1. **Code-based**: which uses the relationship between the code and test cases. It selects test cases based on the changes occur, where two versions of the code - before the change and after it – are required.
2. **Model-based**: which uses the relationship between the model elements and test cases. These elements are traversed to select the test cases that will be used in re-testing. Most of these techniques are based on the UML models, where different UML models are used in this approach.

Some of the code-based approaches are very specific to the programming language used to develop the code [2]. The code-based approach causes problems for software products that are large, complex, and are frequently modified. In addition, it becomes more problematic if different parts of a program are written in different programming languages. Some techniques overcome the drawbacks of the code-based approach by using hybrid approach, which combines between the code-based and the model-based.

Swarnendu Biswas et al [11], proposed a model-based regression test selection technique for embedded programs. They proposed a graph model constructed from program analysis, which captured the different characteristics of embedded programs that were relevant to test selection. It then enhanced the model with information extracted from the SRS document, the analysis and the design models.

Analysis: This technique focused on regression test selection of embedded programs. Moreover, it was applied on small applications.

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

Qurat-ul-ann Farooq et al [12] introduced a state-based selective regression testing methodology for evolving state-based systems along with a tool support named START - an Eclipse-based tool for state-based regression testing compliant with UML 2.1 semantics. The proposed technique used the relationships between the class diagram, state machine and the corresponding test suite to deal with the change propagation. Changes in the class diagram were captured by comparing the baseline version and the delta version of the class diagrams along with class invariants and operation contracts. After capturing the class-driven changes, the authors developed a "StateMachineComparator" tool that compared the baseline and delta version of state machines along with state invariants. Changes in both versions were detected and class-driven changes were used to obtain the affected elements of the state machine. At the end set of affected test, cases were selected using "RegressionTestSelector" tool that traced the state-driven changes to the corresponding test cases.

Analysis: The advantage of this technique is that it reported a tool support for the model-based regression test selection. However, it was applied only on one case study.

Ruchika Malhotra et al, [13] on the other hand, introduced another regression test selection and prioritization technique, which prioritized test cases in test suite and selected from the prioritized test suite. It recommended using high priority test cases first and then low priority test cases and so on until both time and resources are available or a reasonable level of confidence about correctness is achieved. This technique used two main algorithms:

a. A modification algorithm that determines the modified source code and counts the number of modified lines of source code covered by each test case.
b. A deletion algorithm that deletes the number of deleted source code lines from the count of the test case and removes those test cases that cover only those lines that are covered by other test cases of the program.

Analysis: The technique combined both selection and prioritization, which helped in reducing the test cases by a significant number. However, no metrics were used for evaluating the technique, nor for determining how prioritization is done. It was not applied on large size programs, which cannot guarantee the efficiency and scalability of the technique.

In 2012, Chhabi Rani et al [14] introduced a hybrid technique for regression test cases selection for Object-Oriented Programming. This technique was based on both the source code of an object-oriented program and the UML state machine models of the affected classes. The technique constructed a dependency graph model that captured the control, data dependency and model elements affected during the change. The technique also determined the affected methods from the state machine model. It selected for regression the test cases that traverse not only the affected model elements in program, but also the affected methods in the state machine. The technique was compared to Larsen and Harold's [18] which constructed system dependence graphs for object-oriented software by applying efficient slicing algorithms. A system dependence graph consisted of a program dependence graph representing the "main" program either in the system or in a simulation of a calling environment, whereas a class dependence graph represented classes constructed for each class in the system.

Analysis: The technique increased the selection of faults revealing test cases by 27.89% comparing to Larsen an Harold's System Dependence Graph. However, it was applied only to small examples of programs that cannot guarantee its scalability.

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

3.2 Test Cases Reduction Techniques

Test cases reduction techniques focus on finding a minimized set of test cases without compromising fault detection capability. This improves the required testing effort by identifying and eliminating the redundant (or unnecessary) test cases according to some test adequacy criteria.

Dmitry Kichigin [5] introduced a test suite reduction technique for regression testing of simple interactions between two software modules. The technique was based on modelling the behavior of interactions between software modules and used sequences of interface functions invoked during software execution. The module's interactions were modeled using the sequences of module's interface functions invoked during the execution of a program. The parameters passed to the functions were considered as well as their names and the tests that initiate the same sequences of interface functions repeat themselves in the module interactions that they test. The size of reduced test suite, percentage of reduction and fault detection rate metrics were used to evaluate this technique.

Analysis: The technique did not require source code access or instrumentation, as it was based on modelling of interactions "behavior" on a test suite. However, only simple interactions between two modules were considered for this technique, which makes results on complex interactions are not guaranteed.

At 2010, Saeed Parsa et a [6] presented a greedy algorithm for reduction named " Bi-Objective greedy algorithm", which aimed to select a test case that satisfied the maximum number of testing requirements, while having minimum overlap in requirements coverage with other test cases. The objective of this algorithm was to (i) select effective test cases in fault detection, and to (ii) remove redundancy from the test suite. It selected unique test cases in terms of requirements coverage to achieve significant suite size reduction and improved their fault detection effectiveness, then used test case-requirement matrix, which showed the mappings between test cases and testing requirements. The elements consisted of 1's and 0's, which indicated satisfying or dissatisfying the requirements by test cases respectively. The algorithm got the number of test cases that resulted from multiplying the test case-requirement matrix by its transposed matrix. Each diagonal element of this matrix showed the number of unmarked requirements covered by the corresponding test case and each non-diagonal element shows the number of requirements coverage in which test cases overlap. Repeatedly, it selected a test case by adding the test case with the maximum diagonal value into a list, and the test case with the minimal values into another list. It then selected a test case from the intersection of both lists; redundant test cases were removed. .

Analysis: The algorithm was measured using the percentage of suite size reduction and the percentage of fault detection loss metrics. Experiments were conducted on real test suites "Siemens suite" and "The Space program". However, Siemens programs were limited and their faults were well known.

One year later, Chang-ai Sun [15] introduced another test suite reduction method for conservative regression testing. The algorithm constructed test constraints for each previously discovered bug in a Boolean formula which specified the conditions that can guarantee the detection of the targeted bug in a program under test. The algorithm employed program analysis techniques including slicing, chopping and path conditions. Program slicing decomposes programs by analyzing the data flow and control flow [16], and chopping filters slices to know how statements influence each other. Whereas path conditions give necessary conditions under which a transitive dependence

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

between the source and target node exists [17] to obtain trigger conditions and propagation-conditions of the test constraint. The test constraints were merged for the common test constraints between two bugs.

Analysis: The proposed method can be used to guide the generation of high-quality test cases for regression testing. However, no direct metrics were mentioned in the evaluation of this technique for reduction.

3.3 Test Cases Prioritization Techniques

Test cases prioritization is a vital regression testing approach. It sorts existing test cases for regression testing according to certain parameters to attain performance goal.

Adam M. Smith, et al [9], extended algorithms of selection and prioritization to greedily reduce and prioritize the tests by using both test cost (e.g., execution time) and the ratio of code coverage to test cost. They conducted an experimental analysis on eight case study applications to evaluate the effectiveness of extending the following four techniques: Harrold Gupta Soa, Delayed greedy, Traditional greedy and optimal greedy algorithm.

Harrold Gupta Soa (HGS) technique selected a representative set of test cases from a test suite that provided the same coverage as the entire test suite by identifying, and then eliminating the redundant and obsolete test cases in the test suite [18]. The Delayed Greedy (DGR) technique represented a greedy heuristic algorithm. It used coverage information to make intelligent decisions "greedy choices" to select a minimal subset of a test suite T by iteratively exploiting implications among the coverage requirements and the implications among the test cases and the implications among the coverage requirements to cover all the requirements by this test suite T [19]. In 2007 [20] they presented GRD Traditional Greedy, a standard greedy algorithm that iteratively picked a test case that covers maximum unsatisfied branches of the program until all branches were satisfied. In another study, Optimal Greedy (2OPT) algorithm was introduced as a greedy algorithm that performed all-pairs comparisons. It updated coverage information for each unselected test case following the choice of each pair of test cases. These four extended techniques (Harrold Gupta Soa, Delayed greedy, Traditional greedy and Optimal greedy) were evaluated using RFFT (Reduction Factor for Time), RFFS (Reduction Factor for Size), and CE (Coverage Effectiveness).

Analysis: Adam M. Smith's study extended and evaluated already existing algorithms for regression testing but was not tested with large case study applications.

Hyunsook Do, et al [8], introduced a series of experiments conducted to evaluate the effect of time constraints on the costs and benefits of prioritization techniques. They relied on an economic model EVOMO (EVOlution-aware economic Model for regression testing). It is currently the only existing economic model capable of capturing the foregoing factors comprehensively.

Analysis: This paper showed that time constraints can certainly play a significant role in determining both the cost-effectiveness of prioritization techniques, and the relative cost-benefit tradeoffs among techniques.

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

Shin Yoo et al [21], transferred techniques from the regression test literature into industrial practice. They adopted a multi-objective search-based test suite selection technique based on Pareto efficient multi-objective [22] within Google's test environment. The Pareto efficient took multiple objectives such as code coverage, past fault-detection history and execution cost as test selection criteria for prioritization. It seeks test suites that maximize coverage and historical fault revelation, while minimizing execution time.

Analysis: This work highlighted that the optimization of coverage and time alone was insufficient. Additionally, it should use the historical fault revelation in prioritization. It also emphasized the importance of including industry in the research. However, it was applied only on one test environment of Google.

Thillaikarasi Muthusamy et al [23] introduced a new technique for test case prioritization. Their technique assigned six factors for each test case during test design and analysis phases.

The factors assigned to each test case were:

1. Customer-Allotted Priority (CP): which measures the implication of a requisite to the customer, the values of each need were assigned by the customers.
2. Code Implementation Complexity (IC): which is an individual measure of the complexity expected by the development team in implementing the necessity.
3. Changes in requirement (RC): which is a degree assigned by the developer for indicating the number of times a requirement is changed in the development cycle with respect to its origin date.
4. Fault Impact of Requirements (FI): It distinguishes the requirement that had customer reported failures, the number of in-house failures and field failures determine the fault impact of requirements.
5. Completeness (CT): It indicates what is needed as per the requirement for a function to be executed, the rate of success, the limitations to be followed for the function is to be executed and any limitation which manipulates the expected solution for example the boundary constraints.
6. Traceability (TR): It defines whether a requirement is properly tested is cumbersome for evaluators. This c value is determined after assessing individual requirement for the concerned traceability and the standard of software can be improved by opting the traceability of the requirement into consideration is chosen for subsequent usage

Weighted prioritization value (WPV) was then computed for each test case,

$$WPV=\text{Value of factor} * \text{Weight of Factor} \ldots (14)$$

and then the Weighted Priority (WP) was computed. Prioritization was based on WPV and WP of every test case in the test suite. The technique was evaluated using APFD metric.

Analysis: This technique proposed a new practical set of weight factors used in the test case prioritization process. However, it depended on the manual input of all the factors, which may encounter human error and subjective weighting.

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

4. Introducing Data Mining to Regression Testing

There has been an explosion in the use of data mining techniques in the exploration and analysis of large quantities of data in order to discover meaningful patterns and rules. Accordingly, data mining algorithms have been potentially used to solve automation software testing problems with large data. In this section, we discuss the use of data mining techniques to design a minimal set of regression tests or to prioritize the test sets.

Shin Yoo et al [24], introduced a clustering technique in which test cases were clustered based on their similarity of the features tested using the runtime behavior. However, the prioritization between clusters was done using human judgment. The proposed technique represented the execution of each test case by a binary string. Each bit represented a statement in the code. If the test case executed, then this statement's corresponding digit is 1, else it is 0. An agglomerative hierarchical clustering technique was then applied to group the test cases with closer distances using Hamming distance to calculate the distance between test cases binary strings. Prioritization of test cases in the same cluster was done using traditional coverage-based greedy algorithm. However the prioritization of the clusters itself was done using the human tester intervention using APFD metric.

Analysis In this approach, the clustering was applied to reduce the number of pair-wise comparisons making it scalable. However, human involvement was required in the prioritization of the clusters itself which caused human input erroneous and subjective results.

Ryan Carlson et al, [25] conducted empirical studies using industrial software product as Microsoft Dynamics X. They implemented a new prioritization technique with a clustering approach using code coverage, code complexity and history data of real faults. That technique used agglomerative hierarchal clustering method, which merged test cases with closer code coverage similarity. The distance between the cluster and any of the remaining test cases was determined by averaging the distance between each of the elements of the cluster and the test case. A prioritization technique was then applied using the code coverage information, code complexity matrix, the history data of faults and a combined technique using the arithmetic mean of code complexity and fault detection ratio. The test cases were re-ordered in an order that put the highest average value earlier.

Analysis: The technique was evaluated using AFDP metric and was applied on a massive data in real software repositories. However, it was applied only on financial subsystems for Microsoft dynamics Ax programs, where it might not apply on all dynamics Ax programs or other company's products.

Songyu Chen et al [3], introduced a semi-supervised learning technique for regression test selection. This approach introduced a semi-supervised clustering method named semi-supervised K-means (SSKM), which combined SSDR and K-means. SSDR was used as a pre-process of the original data with limited constraint information before K-means. The SSKM was used to group tests into clusters. A popular sampling strategy, namely adaptive sampling strategy, was used to select data from clusters and was evaluated using F-measure.

Analysis : It was the first time to apply the semi-supervised clustering in test selection, or software testing. However, the proposed technique used only a small set of subject programs, modified versions and test sets. In practice, the situations can be challenging for large-scale systems.

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

A year later, Arvind Kumar Upadhyay et al [26], introduced another clustering-based prioritization technique, where test suites were scheduled using clusters. The proposed technique applied the clustering-based prioritization on a quadratic equation problem. It represented the equation using flow graph and then calculated the cyclomatic complexity, which was used to indicate the complexity of a program by directly measuring the number of linearly independent paths through a program's source code. Cycolmatic complexity was used to get the number of independent paths. K-means algorithm was used to group test cases of similar paths in the same cluster, and then prioritization of clusters was done using dendogram method, which is a tree-structured graphical representation of the resulting hierarchy.

Analysis : The technique was evaluated using APFD metric, where it minimized the APFD measure than normal prioritization. However, it targeted very simple test suites and was not applied on complex, or large-scale test suits.

S.Raju et al [10] introduced another cluster-based prioritization technique using certain factors entered by user. The proposed technique received from the user the following values for each test case:

- Rate of Fault detection (RF): The average number of faults for each requirement.
- Requirements Volatility (RV): The number of times a requirement has changed.
- Fault Impact (FI): The fault severity.
- Implementation Complexity (IC): How complex is the implementation of the requirement.

Agglomerative hierarchal clustering was used to group similar test cases in the same cluster. The Prioritization Of Requirements Test (PORT) algorithm was used to prioritize traceability between requirements and test cases, where PORT is a value-driven approach to system-level test cases prioritization, The technique calculated the Prioritization Factor Value (PFV), which measured the importance of testing a requirement by multiplying the factor value for each requirement with the factor weight. Furthermore, the Weighted Prioritized factor (WP) was calculated for each test case from the PFV of the associated requirements. The test cases were ordered for execution based on the descending order of WP values.

Analysis : This algorithm was evaluated using APFD and PTR metrics, where it effectively prioritized the test cases. However, it was applied on one bank application only, besides that, manual input of the factors was required and no prioritization was applied outside a cluster.

Junaid Arafeen et al [27] presented a test case prioritization technique using requirements-based clustering. This technique used text mining in order to determine the clusters of requirements. It used k-means algorithm to cluster similar requirements after using term extraction and creating term-document matrix. After the clustering were formed, the test cases clusters were formed by mapping each test case to its relevant requirements. Prioritization between test cases in the same cluster was done using the source code information, whereas prioritization between clusters was done based on the source code information along with prioritized requests from the client producing re-ordered test cases.

Analysis : The technique was evaluated using AFDP. The use of requirements-based clustering approach, which incorporated traditional code analysis information, improved the effectiveness of test case prioritization techniques. However, no analysis was explained for choosing the different metrics for code complexity and changing the number of clusters in the algorithm. In addition, it was applied

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

on small and medium size programs. Accordingly, results from their study cannot be interpreted in the context of industrial applications.

5. Regression testing tools in market

With the increasing need for regression testing tools in the market, many applications have recently emerged to automate the regression testing process. Most of the tools work with the re-test all approach by re-running all the test cases in the regression phase. In addition, there are no specific illustrations mentioned for the scalability of these tools in dealing with large-scale systems. In the below table, we present the most famous regression testing automation tools in market along with their advantages and disadvantages.

Table 1 : Regression testing tools

Tool Name	Description	Advantages	Disadvantages
Test Complete [1]	A test platform for easily constructing, maintaining, and executing automated tests for desktop, web, mobile, and client-server software applications. It has many features needed to make regression testing fully automated by re-running automated functional tests	An open-source tool that performs functional tests and it is also possible to develop tests with scripts.	No algorithm used for selecting or prioritizing test cases, only re-running all the test cases for regression is used.
Rational Functional Tester[2]	A java tool used to automate the test cases of software applications. This is primarily used for automating regression test cases.	Provides testers with automated capabilities for data-driven and keyword testing.	No algorithm used for selecting or prioritizing test cases, only re-running all the test cases for regression is used.
Silk Test[3]	An automated testing tool for regression testing, supporting multiple technologies like AJAX, Web 2.0, .NET and JAVA, Silk Test boosts productivity	It contains all the source script files and supports object oriented implementation.	It uses the proprietary 4Test language for automation scripting which is not familiar and uses the re-test all approach
Junit [4]	A unit-testing framework for the Java programming language. It is a simple framework to write repeatable tests	It is an open-source framework, providing graphical user interface to	It was originally intended and still primarily used for "unit testing" —

[1]http://smartbear.com/products/testcomplete/
[2] http://www-03.ibm.com/software/products/en/functional
[3] http://www.borland.com/products/silktest
[4]http://junit.org

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

		write and test source code quickly and easily.	that is, white box testing of individual components outside the context of the whole system, not regression testing.
Q Engine [5]	An automated testing platform to test the functionality and performance of web applications and web services in Windows and Linux platforms. It provides a web-based interface and an integrated test management solution that enables test automation engineers to schedule test suites for unattended test execution, and run test suites from command line mode using batch/shell files for regression testing	Tool is developed using Java, which facilitates portability and multiple platform support.	Used only for web applications and web services and uses the re-test all approach.
Mercury Win Runner [6]	A tool for enterprise-wide functional and regression testing. Fully integrated with the HP Business Process Testing Solution. It captures, verifies, and replays user interactions automatically.	It implemented a proprietary Test Script Language (TSL) that allowed customization and parameterization of user input.	It is no longer supported by HP, thus might not be a good choice of a tool for someone starting out in implementing an automated testing structure
Scout [28]	A tool developed by Microsoft which codified a greedy algorithm for selecting tests, vindicating the minimization approach	It uses an algorithm for selecting test cases, which significantly reduce the number of regression tests being re-run.	Not widely used in market.
Test Impact Analysis[7]	It is a feature in Visual Studio 2010 (Premium and Ultimate editions) that analyzes the changes made to the code base. It also determines what unit tests may be affected, or "impacted" by the changes of the code. The	It not re-runs all the test cases in regression, but only that affects the code changes.	Native code is not yet supported, it does not select a minimal set of tests to re-run, but instead, It selects an aggregate set,

[5] https://www.manageengine.com/products/qengine
[6] http://www.starbase.co.uk/what-we-sell/hp/functional-testing/winrunner.html
[7] http://visualstudiomagazine.com/articles/2011/02/10/test-impact-analysis.aspx

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

	developer has the option to run only the impacted tests, in effect testing just the code changes that were made, or running all the unit tests in the solution to perform full regression testing		which includes all tests traversing the modified code.
QTP[8]	An automated software designed to automate functional and regression test. It compares the actual and expected result and reports the results in the execution summary.	Developing automated tests using VBScript does not require a highly skilled coder, Easy to use. It can also be used for mobile application testing.	Works in Windows operating system only, not all versions of browsers are supported. The licensing cost is very high, and the execution time is relatively higher as it puts load on CPU & RAM.
Watir[9]	Uses the family of Ruby libraries for automating web browsers. It allows the user to write test cases that are easy to read and maintain	An open source supports all major browsers. Open-source Framework.	Test scripts are written in the Ruby programming language, which is unfamiliar, needs skilled programmer, and uses the re-test all approach.
TOSCA[10]	Used for automated execution of functional and regression software testing. In addition to test automation functions, TOSCA includes integrated test management, a graphical user interface (GUI), a command line interface (CLI) and an application programming interface (API).	Very rich automation/test management framework	Consider the coding needed to integrate with your application

6. Regression Testing Challenges and Comparative Analysis.

Although regression testing has taken huge efforts in research, there are still many challenges and gaps in adopting regression testing methodologies. Some of the challenges were that no sufficient

[8] http://www.quicklearnqtp.com/2009/07/regression-testing-framework-in-qtp.html
[9] http://www.watir.com
[10] http://www.qualitytesting.info/page/tosca-testsuite

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

number of experiments and trials are considered in many experiments, no data sets are created containing information about the efficiency and effectiveness of the studies. The testing coverage report and testing results of the presented algorithms are not mentioned in details in the papers [29] In addition, there is a challenge related to the lack of tool support that helps to select, remove, and re-order the tests within real-life application. Some other researches mentioned the challenge of scalability issue in reality, the weakness of most approaches that they did not scale well or were not studied with large data sets. Others did not show a worthy performance with large sets.

With continuous regression runs, test suites become large; the entire regression test suite cannot be executed due to time and budget constraints [30]. Minimizing test suite with maximum test coverage achievement remains a challenge. The above challenges and gap between the research and practice results in the limitation of the industrial adoption of the current regression testing techniques.

7. Proposed Approach

The purpose of our proposed model is to provide a fully automated regression-testing tool that takes the full test cases set, the traceability matrix showing relations between the test cases and other components like code reference and the list of the code modules that were changed during different phases of the project, and promotes to the user certain test cases to be used for regression

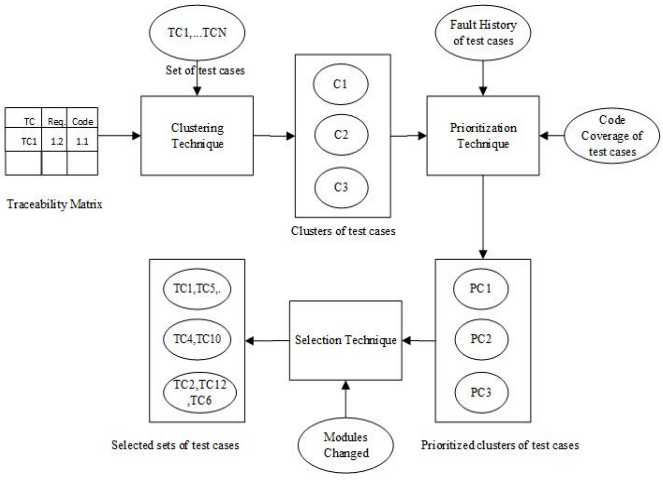

Figure 4 : Our Proposed Model

7.1 Clustering

In order to support scalability, all test cases are clustered based on their code coverage similarity [31]. The test cases covering similar code modules are in the same cluster. Each test case is defined by a string of 0s and 1s, where each bit in the string represents a code module; 1 means that this module is covered by this test case and 0 means this module is not covered by this test case.

Distance between the defined strings of test cases is calculated using the Hamming distance [32]. Using clustering technique, close test cases (with minimum hamming distance difference) are grouped in the same cluster, indicating that they test similar modules.

International **J**ournal of **S**oft **C**omputing and **S**oftware **E**ngineering (**JSCSE**)
Vol.5, No.4, 2015

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

7.2 Prioritization

Prioritization of clusters is then applied based on the code coverage and the fault detection history. The code coverage of test cases in the same cluster represents the number of modules covered by all the test cases in the cluster. Whereas, the fault history for the test cases in each cluster represents how many test cases in each cluster have failed before. Each cluster is ranked based on the above numbers. The cluster containing test cases with the maximum code coverage and maximum number of failed test cases takes the highest rank, and thus it is given the highest priority.

7.3 Selection

Eventually, selection of test cases from each cluster is done to run the regression testing instead of the whole list. In the selection phase, the list of modules changed are used combined with the priority list of clusters. For each cluster, the priority of the cluster and the number of changed modules covered by the test cases in the cluster are checked. Based on the calculation above, a percentage of the test cases are selected from each cluster according to a certain threshold, where this threshold is determined through a learning process to avoid human errors and subjective input.

8. Conclusion

In this paper, an intensive study has been presented on the various regression test selection, reduction and prioritization techniques. Advantages and limitations of the proposed techniques were also discussed, An emphasize to the use of data mining techniques into test cases selection and prioritization of regression testing has also been investigated to address scalability-related issues. In addition, we elaborated the parameters of evaluation used in the different techniques, besides the tools used for regression testing in the market. Along our research, we have analyzed the challenges and gaps that are still found in the presented techniques of regression testing. Finally, a new approach for regression testing has been proposed to tackle large-scale systems limitations. This approach combines clustering along with both prioritization and selection techniques with no manual input required from the user, taking into consideration the fault history and code coverage of test cases. This helps to achieve more efficient regression testing in terms of cost and avoid human erroneous.

References

[1] Trivedi, S. H., Software Testing Techniques, International Journal of Advanced Research in Computer Science and Software Engineering ,vol. 2 no.10, pp.433–439,2012

[2] Bharati, C., & Verma, Analysis of Different Regression Testing Approaches. International Journal of Advanced Research in Computer and Communication Engineering,vol. 2 no.5, 2150–2155,2013

[3] Chen, S., Chen, Z., Zhao, Z., Xu, B., & Feng, Y, Using semi-supervised clustering to improve regression test selection techniques. Fourth IEEE International Conference on Software Testing, Verification and Validation , pp. 1–10,2011

[4] Ur, S., Khan, R., Lee, S., Parizi, R. M., & Elahi, M., An Analysis of the Code Coverage-based Greedy Algorithms for Test Suite Reduction, The Second International Conference on Informatics Engineering & Information Science (ICIEIS2013), pp.370–377,2-13

[5] Kichigin, D. Test Suite Reduction for Regression Testing of Simple Interactions between Two Software Modules, Perspectives of Systems Informatics ,pp. 107–123,2010

[6] Parsa, S., & Khalilian, A., On the Optimization Approach towards Test Suite Minimization. International Journal of Software Engineering and Its Applications,vol.4 ,no. 1, pp.15–28,2010

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

[7] Yoo, S., & Harman, M, .Regression Testing Minimisation, Selection and Prioritisation - A Survey, 2009,Software Testing Verification and Reliability, 22(2) , pp. 67 - 120, 2012.
[8] Do, H. D. H., Mirarab, S., Tahvildari, L., & Rothermel, G, The Effects of Time Constraints on Test Case Prioritization: A Series of Controlled Experiments. IEEE Transactions on Software Engineering, vol. 36, no.5,2010
[9] Smith, A. M., & Kapfhammer, G. M,An empirical study of incorporating cost into test suite reduction and prioritization, Proceedings of the 2009 ACM Symposium on Applied Computing - SAC '09, 461,2009
[10] S. Raju and G.V. Uma , An Efficient Method to Achieve Effective Test Case Prioritization in Regression Testing using Prioritization Factors . Asian Journal of Information Technology,vol. 11: pp. 169-180,2012
[11] Biswas, S., Mall, R., Satpathy, M., & Sukumaran, S., A model-based regression test selection approach for embedded applications. ACM SIGSOFT Software Engineering Notes, vol. 34 no. 4, pp. 1,2009
[12] Farooq, Q., Iqbal, M. Z. Z., Malik, Z. I., & Riebisch, M, A Model-Based Regression Testing Approach for Evolving Software Systems with Flexible Tool Support. 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems, pp.41–49,2010
[13] Malhotra, R., Kaur, A., & Singh, Y. ,A Regression Test Selection and Prioritization Technique, Journal of Information Processing Systems, vol 6,no 2,2010
[14] Panigrahi, C. R., & Mall, R. , A Hybrid Regression Test Selection Technique for Object-Oriented Programs. International Journal of Software Engineering and Its Applications, 2012, vol. 6,no.4) , pp17–34.
[15] Sun, C., A Constraint-based Test Suite Reduction Method for Conservative Regression Testing. Journal of Software, vol. 6 , no.2, pp.314–321,2011.
[16] Weiser, M. , Program Slicing, IEEE Transactions on Software Engineering, SE-, vol.10, no.4, pp.352–357,1984
[17] Krinke, J. Slicing , Chopping , and Path Conditions with Barriers, Software Quality Control ,vol. 12, no.(4) ,Pages 339 - 360,2004
[18] Clemson, M., Gupta, R., Jean, M., & Unwersity,A methodology for Controlling the Size of a test suite. ACM TransactIons on Software Engineering and Methodology, vol2 , no. 3, pp.270–285,1993
[19] Tallam, S., & Gupta, A concept analysis inspired greedy algorithm for test suite minimization ,The 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering - PASTE '05, 35, 2005
[20] Li, Z., Harman, M., & Hierons, R. M., Search Algorithms for Regression Test Case Prioritisation, Software Engineering, IEEE Transactions on Software Engineering ,vol. 33, no.4, PP 225 - 237, 2007
[21] Nilsson, R., & Harman, M., Faster Fault Finding at Google Using Multi Objective Regression Test Optimisation,2011
[22] Yoo, S., & Harman, M., Pareto Efficient Multi-Objective Test Case Selection, ISSTA '07, London, U.K,2007
[23] Muthusamy, T., A New Effective Test Case Prioritization for Regression Testing based on Prioritization Algorithm. International Journal of Applied Information Systems, vol. 6, no.7 , pp.21-26, 2014
[24] Yoo, S., Harman, M., Tonella, P., & Susi, A. ,Clustering Test Cases to Achieve Effective & Scalable Prioritisation Incorporating Expert Knowledge, ISSTA'09 Proceedings of the eighteenth international symposium on Software testing and analysis, pp. 201-212,2009
[25] Carlson, R. , Hyunsook Do ; Denton, A. , A Clustering Approach to improving test case prioritization.27th IEEE International Conference on Software Maintenance, pp 382-391, 2011

Published online: April 25, 2015

e-ISSN: 2251-7545
DOI: 10.7321/jscse.v5.n4.1

[26] Upadhyay, A. K., & Misra, A. K. , Prioritizing Test Suites Using Clustering Approach in Software Testing, International Journal of Soft Computing and Engineering (IJSCE) no. 4, pp. 222–226,2012
[27] Arafeen, M. J., & Do, H, Test case prioritization using requirements-based clustering. Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation, ICST , pp.312–321,2013
[28] Hartmann, J., 30 Years of Regression Testing : Past, Present and Future. PNSQC 2012 Proceedings, pp.1–8,2012
[29] Kapfhammer, G. M., Empirically Evaluating Regression Testing Techniques: Challenges, Solutions, and a Potential Way Forward, IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, pp.99–102,2011
[30] Engström, E., Runeson, P., & Skoglund, M., A systematic review on regression test selection techniques. Information and Software Technology, vol. 52, no.1, pp.14–30, 2010
[31] Berkhin, P. Survey of clustering data mining techniques.Accrue Software, Inc 2002.
[32] He, M. X., Petoukhov, S. V., & Ricci, P. E. Genetic code, Hamming distance and stochastic matrices. Bulletin of mathematical biology,Vol 66,no.5,pp. 1405-1421, 2004