History Based Multi Objective Test Suite Prioritization in Regression Testing Using Genetic Algorithm

Megala.T
Computer science and Engineering
Pondicherry Engineering College
Pondicherry, India
megelaganesh@pec.edu

K.Vivekanadan
Computer Science and Engineering
Pondicherry Engineering College
Pondicherry, India
k.vivekanandan@pec.edu

Abstract—Regression testing is the most essential and expensive testing activity which occurs throughout the software development life cycle. As Regression testing requires executions of many test cases it imposes the necessity of test case prioritization process to reduce the resource constraint. Test case prioritization technique schedule the test case in an order that increase the chance of early fault detection. In this paper we propose a genetic algorithm based prioritization technique which uses the historical information of system level test cases to prioritize test cases to detect most severe faults early. In addition the proposed approach also calculates weight factor for each requirement to achieve customer satisfaction and to improve the rate of severe fault detection. To validate the proposed approach we performed controlled experiments over industry projects which proved the proposed approach effectiveness in terms of average percentage of fault detected.

***Keywords**- Regression Testing; Prioritization; Genetic Algorithm; Fault Severity*

*****

I. Introduction

Software testing is a process of finding errors or bugs in a developed software product. Around 50% of total software cost and time is spent in software testing [1].Regression testing is one of the software testing activities predominately performed when changes are done to the existing software product. It also includes activities such as enhancement of a software product, error correction, and optimization .The objective behind regression testing is to ensure the quality of the software product, in a way that the newly introduced change to the software product do not deteriorate the working of the existing software product. In general “reuse all” test case is the method adopted in Regression testing which is practically not possible. For example a product consisting of 20,000 lines of code, running entire test suite may require eight weeks [2] which may consumes resources such as time, cost and manpower. Therefore to overcome this, many researchers have proposed several regression testing techniques such as test suite minimization, selection and prioritization technique. Among these technique the prioritization technique is most effective were test cases are not discarded and the tester can schedule the test cases in an order depending upon the objective and budget situation.

In existing there are number of prioritization approach which utilizes only source code information (statement, functional and conditional)in which prioritization cannot be applied if source code is unavailable and very few technique based on other artefacts such as system requirement. Another issue is during regression testing a huge amount of historical data are constantly stored in database which are ignored and not effectively used by many of existing techniques.[3].Based on our understanding another limitation is, in many cases faults can be critical which causes customer significant money loss, deferring product shipment ,or having safety implication. Therefore most severe faults can be detected earlier by incorporating requirement weightage also through which customer perceived quality can be achieved. Since the regression testing problem is a NP-hard problem. Meta-heuristic algorithm are employed to solve prioritization problem. Genetic algorithm is an evolutionary algorithm which is widely used to solve complex problem due to its customization property and some researchers also suggest to employ genetic algorithm in test case prioritization [4]. To address the above mentioned limitation in this paper we propose a history based prioritization technique which uses genetic algorithm to prioritize test case based on weight factor, requirement coverage and fault identified with severity

The rest of the paper is organized as follow: Section 2 discuss the related works on prioritization technique. Section 3 presents the proposed approach and the empirical evaluation and there outcomes are discussed in section 4.In section 5 conclusion and future work is presented.

II. Background and Related Works

In this section, we discuss the test case prioritization techniques and their related works and also various background information on regression testing and genetic algorithm.

A. Regression Testing

Regression testing is a kind of software testing that revalidates a software system in order to find out whether the modification to the software system cause any miscue or not, among the several versions of the software system. As complete regression testing is an expensive process, several techniques have been examined for a more efficient regression testing. The four main techniques for regression testing are test case selection, test case reduction, retest all and test case prioritization

- Retest all: Retest all is a regression testing technique in which all the test cases in the test suite are executed.
- Test case Reduction: In this technique the test suite is reduced. For instance the redundant test cases are discarded
- Test case Selection: In test case selection, a subset of test cases are selected from entire test suite.
- Test case Prioritization: In Prioritization, the test cases are scheduled based on user objective

Among the four technique, test case prioritization technique is the most effective technique [3].

B. *Test case prioritization problem*

Test case prioritization technique reorder the test cases in a test suite in order to achieve the desired objective. When historical information of test cases are taken for prioritization it is called as history based prioritization of test cases.

Problem definition: In history based test case prioritization given: T, a test suite: PT, the set of all permutation of T;HI, the historical information of all element in PT; F, a function from PT to real numbers; and ft ,the fitness function from HI to real numbers.

Problem:

Find $T_{Max} \in PT \text{ such that } \forall T' \in PT', (T' \neq T_{max}),$

$$[ft(HI_{T_{max}}) \geq ft(HI_{T'}) \text{ and } [[f(T_{Max}) \geq f(T')] \quad (1)$$

C. *Related works*

Recently many researchers have proposed various diverse methods on test case prioritization. Many techniques prioritize test case based on analyses of source code with different testing adequacy criteria such as functional, statement and conditional coverage [5]. The limitation of this method is the tester cannot prioritize test case without source code. In order to overcome the above limitation prioritization approaches based software artifacts such as requirement based prioritization, cost-cognizant and historical information based prioritization was proposed and studied in [6, 9]. Xialin et.al proposed a history based prioritization approaches which uses historical value for test case prioritization [3]. Rothermel et al and Walcott et al also suggest to apply genetic algorithm in test case prioritization [9].Yuchi et al proposed history based cost cognizant prioritization which uses genetic algorithm for producing order [8] Wolcott et al [10] developed a prioritization approach which considered testing time budget.

Very few prioritization technique also uses other artefacts such as requirement specification. S.A, Mary et al [7] presented a model that prioritize test case based on requirement with six factors which emphasis on user perceived testing. Arafeen et al [11] presented a requirement – based clustering technique for prioritization and Srikanth et al proposed a requirement based prioritization called (PORT).M.yoon et.al proposed a model to analyze risk object and prioritize test case by estimating the requirement of risk exposure value [12]. However the majority of prioritization technique does not give importance to user perceived quality testing and effectively use the historical data generated in previous testing. Therefore, this paper presents a multi-faceted prioritization technique which use customer priority and history based approach to prioritize test case.

III. RESEARCH METHOLOGY

In this section, the proposed multi-faceted prioritization technique is described. Also gives an overview of the proposed approach which includes the method for calculating weightage of each requirement and working of genetic algorithm is also illustrated.

A. *Proposed Prioritization Approach*

Problem definition: Given: a test suite T. PT, the set of all permutation of T; HI, the historical information of all element in PT; *F*, a function from PT to real numbers; and *ft*, fitness function from HI to real numbers.

Problem:

Find $T_{Max} \in PT \text{ such that } \forall T' \in PT', (T' \neq T_{max}),$

$$[ft(HI_{T_{max}}) \geq ft(HI_{T'}) \text{ and } [[f(T_{Max}) \geq f(T')] \quad (2)$$

In eq. (2), the function *ft* calculates the value of the ordered based on its historical record. And the function *F* measures the effectiveness of the order.

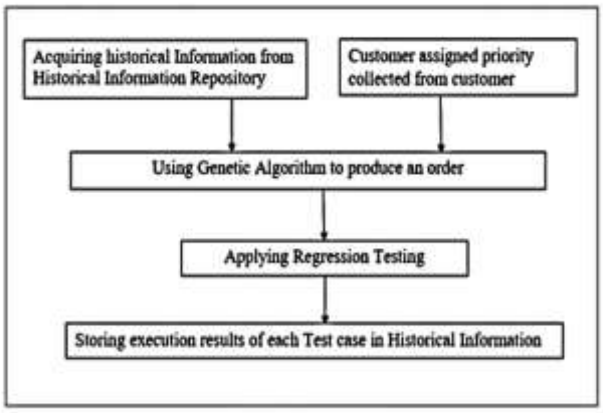
Acquiring historical Information from Historical Information Repository
Customer assigned priority collected from customer
Using Genetic Algorithm to produce an order
Applying Regression Testing
Storing execution results of each Test case in Historical Information

Figure 1.Architecture Diagram

The objective of the given problem is to find the priority of given test case using historical information and schedule the test case in an order which increase the rate of fault detection. As the defined problem is NP-hard, we utilized genetic algorithm in proposed prioritization technique. Since GA is a heuristic technique to produce optimal result, we input information about test case such as fault detected by test case with severity, requirement covered with priority for each requirement from historical information repository. The proposed system architecture is shown in fig. 1.

B. *Proposed Genetic Algorithm*

The proposed genetic algorithm is shown in fig 3, is used to schedule the test case based on historical information. As we seen on step 1 in figure. 3, the genetic algorithm fetches all the information about the test cases such as fault detected *fd*, fault severity *fs*, requirement priority *Rp* (both customer and developer) from historical information repository. The information about test cases are recorded from previous testing activities. Genetic algorithm initiate its process by creating a set of population randomly as shown in step 2 of figure .3. The population consist of chromosome with different test case order. The chromosomes can encoded using different encoding scheme in GA. Since the proposed problem is an ordering problem permutation encoding scheme is used to represent the chromosome. The generated chromosome represent the order of test case execution and the each gene in chromosome represent test case id, an example of chromosome is shown in Fig 2. The chromosome A represent the test case order T6-T4-T5- T1- T3- T2 and chromosome B represent the test case order T3-T6- T4- T5- T2- T1.

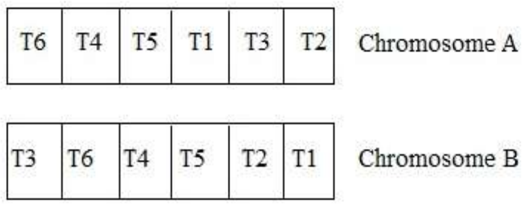

Figure 2.Chromosome Representation

In step 3 of proposed genetic algorithm for loop is the beginning of evolution process the algorithm calculates the fitness value of each chromosome through fitness function and stores the values. Step 5 selects two chromosomes with best fitness value to form the new population by applying recombination operation through crossover and mutation operator .the crossover is applied on selected chromosome to produce chromosome with new test case order. Mutation is the process of creating a random changes on chromosome produced by crossover operation .the crossover and mutation operator are briefly explained in section *D* and **E**. the recombination process is continued until the termination criteria is met. The two termination criteria is to run the algorithm until maximum generation is reached or when test case order with highest fitness is obtained.

```
Algorithm Test Case Prioritization

Input:
T: Test Suite
P:  Size of the population
N: Number of generation
Cp: crossover probability
Mp: mutation probability

Output:
$T_{order}$ :  A test order with highest fitness value is finally generated.

Begin
  Step 1: Fetch information test case cost tc, fault
            detected fd, fault severity fs, requirement
            priority Rp from historical information
            repository.
  Step 2: Generate initial population $P_i \leftarrow$Generate
            population (T, $P_i$, tc, fd, fs, Rp)
  Step 3: For i=1 to n
  Step 4: $F_i \leftarrow$ Evaluate Fitness ($P_i$, tc, fd, fs, Rp)
  Step 5: Select best two chromosomes based on fitness
           value of the chromosome.
  Step 6:  Apply Crossover Operator
  Step 7:  Apply Mutation Operator
End For
Return Test order with highest fitness value
End
```

Figure 3.Proposed Genetic Algorithm for test case prioritization

C. *Fitness Function*

Fitness function evaluates the fitness value of each chromosome in the population .since the aim of the proposed system is to explore an order from existing test suite that has a greatest effectiveness in early fault detection with customer perceived quality. The fitness function *f* is defined and shown in eq 2. The function *f* uses the historical information of test cases and customer assigned priority of requirement to derive the fitness value.

$$f(individual) = \frac{\sum_{i=1}^{n}\left(\sum_{j=1}^{x} fs_j \ * \ \sum_{k=1}^{y} rp_k\right)}{\sum_{j=1}^{f} fs_j \ * \ \sum_{k=1}^{r} rp_k} \qquad (3)$$

Where n is total number of test cases or size of chromosome, f is total number of faults, r is total number of requirements, x is the list of faults explored so far, y is list of requirements covered so far, fs_j is the severity of fault j and rp_k is the priority value of requirement k.

```
Algorithm crossover (parent 1, parent 2, CP)
  Input: Select two chromosome as Parent 1, Parent 2
         from current population
 Output: offspring1, offspring2
Begin
   Generate random number rnd from (0 to 100)
   Check if (rnd<cp)
       Select two crossover point
         P1←crossover point (parent 1)
         P2←crossover point (parent 2)
         Segment1←fragment (p1, Parent1)
         Segment2←fragment (p2, Parent2)
         Offspring1←merge(Segment1,parent2)
         Offspring2←merge(Segment2,parent1)
      Else
         Offspring1←Parent1
         Offspring2←Parent2
   Return
      Offspring1, offspring2
End
```

Figure 4.Algorithm for Crossover Operator

D. *Crossover*

To generate different testing order, recombination operators are applied which produces chromosome of variant. The two type of recombination operators applied are crossover and mutation .Crossover is a recombination operator which generates a new offspring by selecting two chromosome as a parent chromosome from current population and combines segment of one chromosome with segment of another chromosome to produce a new offspring. The generated chromosome inherits the characteristic of both the parent chromosome. A single point crossover is applied to produce different testing order the algorithm for single point crossover is shown in fig.4

Initially the algorithm select two parent chromosome Parent 1 and Parent 2 and a random number (rnd) is generated which ranges from 0 to 100, if the generated r is lesser than the user provided crossover probability value, then the recombination process begin by selecting crossover point P1 from parent 1 and P2 from parent2 chromosome. The sequence before the crossover point from both the parents are copied and the function merge generate a new offspring by joining the copied subsequence of one parent to another by which two offspring are produced. Otherwise if the generated random number is greater than Cp value, the parent individual is unchanged and await for next step.

E. *Mutation*

The mutation operator is applied on chromosome to maintain the diversity in newly generated population.

```
Algorithm Mutation (Offspring_c, MP)

Input: offspring produced by crossover
Output: new Offspring_m.

Begin
    Generate random number rnd from (0 to 100)
    Check if (rnd<Mp)
        Select two mutation point (P1, P2)
        Offspring1←SwitchPosition of genes (P1, P2)
     Else
        Offspring_m← Offspring_c
   End if
  Return
      Offspring_m
End
```

Figure 5.Algorithm for Mutation Operator

In the proposed algorithm random mutation is applied to chromosome and the algorithm for random mutation in given in figure 5.The mutation process begins with generating a random number (rnd) which takes value from 0 to 100.if the random number generated is lesser than user provided mutation probability mp, the mutation process begins with selection of two mutation point and switches the genes within the mutation point which result in generation of new offspring. Otherwise if the value of random number is greater the mutation process is terminated .When mutation is applied to test case prioritization technique it gives chances to test case with less fault detecting capability in current testing but may reveal more fault in future regression test.

F. *Requirement weightage*

Weightage for requirement is calculated based on two factors such as customer assigned priority (CP) and developer assigned priority (DP) and stored in historical repository.

- Customer Assigned Priority (CP): It is the measure of importance to customer requirement or need. The customer assign weightage value ranging from 1 to 10 based on importance, where 10 indicated highly important and 1 indicates less important. The reason behind incorporating CP is to test early the requirement with highest priority which improve the customer perceived value and satisfaction.

- Developer Assigned Priority (DP): It is the measure of complexity of each requirement during implementation. The value ranges from 10 to 1 and obtained from developer and domain expert. The reason to choose DP is the probability of occurrence of fault increase directly when the complexity increases.

Requirement ID	***Customer Assigned priority(CP)***	***Developer Assigned priority(DP)***

Requirement -ID-1	7	6
Requirement -ID-2	8	8
Requirement -ID-3	3	5
Requirement -ID-4	5	4

Table 1: Example of requirement priority value

G. *Fault severity*

Each fault is assigned a severity value with scaling of (1-10) and four classes of severity is defined.

- Severity 1: Severity 1 belongs to failures which are highly severe in nature. The user can no longer use the product until the failure is fixed and severity value of 10 is assigned.
- Severity 2: Severity 2 belongs to failure which are at medium level, this failure happens when there is issue with the product but the usage of the product is not affected in anyway. The issue can be fixed in later releases .For severity 2 failures, a severity value of 8 is assigned.
- Severity 3: Severity 3 is allocated to a failure for which a bug fixing can be completed in later versions and the product can be used with the workaround for the failure. An SV of 4 is assigned to failures with severity 3.
- Severity 4: Severity 4 belongs to least failure level .The failure can be fixed out in later version or not at all carried yet the product can still be used. An SV of 2 is assigned to failures with severity 4.

IV. Experimental Comparision and Evaluation Results

In this section the experimental results of the proposed technique is explained. To investigate the performance of our proposed history based prioritization technique, we perform empirical evaluation in terms of following research question.

Research question 1: Is the proposed technique is effective than other test case prioritization technique

Research question 2: Is the proposed technique give importance to user perceived quality testing.

To evaluate the proposed prioritization technique, a web application project for hospital management is used, which approximately contain 50000 lines of java code developed by version square private .ltd. For the experiment five regression testing is applied. The system level test cases is generated by testing team. And 60 requirement specifications were summarized. The tester collects the requirement weightage from customers and also from developer with the scaling of (10 to 1).based on the data collected and historical data available the prioritization is done .as historical data is not available for first regression testing we use random testing. The parameter used for the proposed genetic algorithm is presented in table 2.

Parameters	
Size of the Population	200
Generations	500
Crossover Rate	1
Mutation Rate	0.7
Test Adequacy Criteria	Requirement Coverage

Table 2: Example of requirement priority value

A. Evaluation Metrics

To evaluate the performance of the proposed approach the average percentage of fault detected metric is chosen.
APFD: Average percentage of fault detected is a metric used for evaluating test cost prioritization technique coined by Rothermel et.al [1].The metric measures how quickly the faults are revealed during the regression testing .The basic formula for APFD is given below.

$$APFD = 1 - \frac{TF_1 + TF_2 + TF_m}{nm} + \frac{1}{2n}$$

In above equation, m is the number of existing fault, and n is the number of test cases .TF_i is the place of test case in prioritized test suite which reveals the fault initially. When higher the APFD value the better the technique performs in terms of fault detection.

B. Results and Analysis

Research Question 1: To address the first research question, the proposed technique performance is compared with other technique in term of early rate of fault detection. The other techniques used for comparison is Random order, Initial order and Reverse order. The same experimental setup is applied for all four techniques. The APFD value for all the technique is given in table.3. Fig 6.shows the APFD distribution of proposed technique versus random order for five regression testing. The y axis indicates the version and x-axis indicates the APFD value from the Fig .6 it can be concluded that the proposed technique has higher APFD than the random order. As version 1 is a first regression testing random method is applied.

Techniques	***V1***	***V2***	***V3***	***V4***	***V5***
Random order	73.36	80.4	82.3	79.9	72
No order	62.3	69	71	64.32	70
Reverse Order	54.6	63	58.9	71	60.02
Hist-GA	75	96.4	98	97	98.8

Table 3: APFD Value for Prioritization Techniques

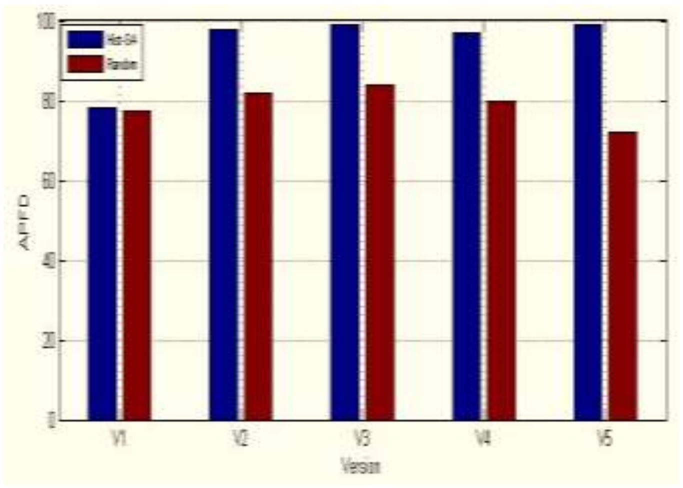

Figure 6.APFD Comparison with Random Technique

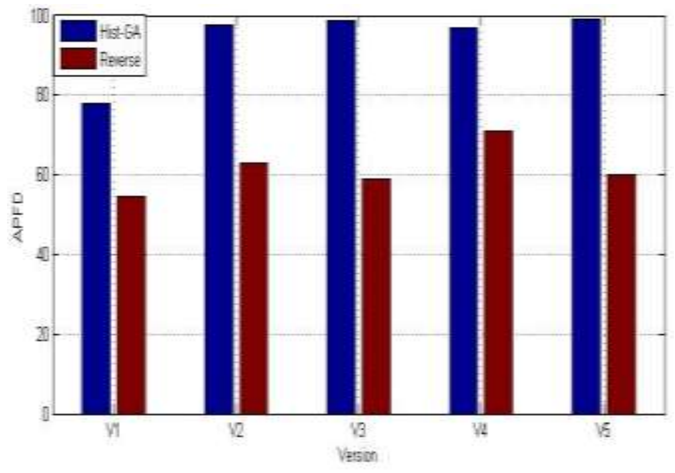

Figure 7.APFD Comparison with Reverse Technique

Fig. 7 shows the APFD comparison of proposed technique and reverse order which shows the proposed technique outperforms than reverse technique. And in Fig. 8. The APFD distribution of proposed technique and no order technique is compared and from the inference the proposed technique perform better than all the three techniques

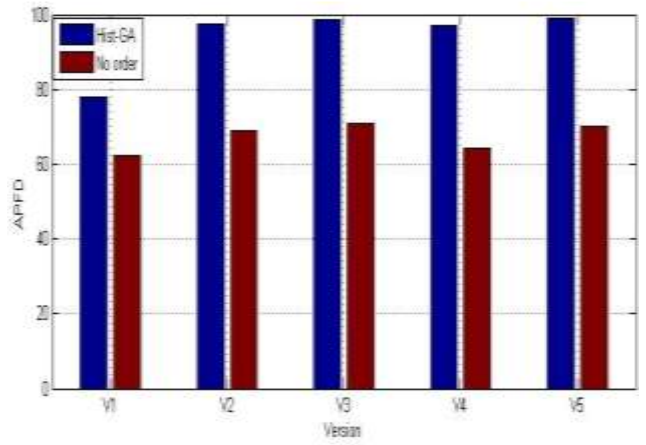

Figure 8. APFD Comparison with No Order Technique

Research Question 2: Is the proposed technique give importance to user perceived quality testing.

The proposed prioritization technique utilizes historical information about the test case to generate test case order along with it also incorporates customer priority for requirement .thus the requirement with highest customer priority are tested early and the faults are detected earlier which helped to accomplish user perceived quality testing. Many of the existing test case prioritization technique fails to provide importance to user perceived quality testing.

V. CONCLUSION

In this paper a new prioritization technique is proposed that orders the test case based on the historical information. The proposed technique prioritize test case based on fault detected, severity of detected fault and priority of requirement without analyzing the source code. The proposed technique is validated by using an industrial project with more than 50000 LOC. In comparison with other traditional techniques random, reverse and no order the proposed technique has a higher efficiency in terms of rate of fault detection. From the experimental result, we conclude that the proposed technique is highly effective and produce an effective test case order that reveals the severe faults early. The proposed technique also includes the priority of customer which helps to achieve user perceived quality testing. Lastly, there are still some research issues such as relating history based model with coverage techniques and resource constrained regression testing which is taken as future work.

REFERENCES

[1] G. Rotherrmel, Untch C. Chu, M. Harrold, Test case prioritization, IEEE Transactions on Software Engineering 27 (10) (2001) 929–948.

[2] H. Srikanth, Requirement based test case prioritization, in: Student Research Forum in 12th ACM SIGSOFT Internationa1 Symposium on the Foundations of Software Engineering, Newport Beach, California, 2004.

[3] Xialin wang, Hongwel Zeng, History Based Dynamic Test Case Prioritization for Requirement Properties in Regression Testing, International workshop on continuous software evolution and delivery (2016), pp.41-47.

[4] Huang Y C, Peng K L, Huang C Y. A history-based cost-cognizant test case prioritization technique in regression testing. Journal of Systems and Software, 2012, 85(3): 626–637

[5] S.Elbaum, A. Malishevsky, G. Rothermel, 'Incorporating varying test costs and fault severities into test case prioritization, in: Proceedings of the 23rd International Conference on Software Engineering, Ontario, Canada, May 2001,pp. 329–338

[6] S. Elbaum, A. Malishevsky, G. Rothermel, Prioritizing test cases for regression Testing, in: Proceedings of the ACM International Symposium on Software Testing and Analysis 25 (5) 2000, pp. 102–112.

[7] R.Krishnamoorthi, S.A.Sahaaya Arul Mary, Factor oriented requirement coverage based system test case prioritization of new and regression test cases, Inf.Softw.Technol.51 (4) (2009)799–808.

[8] Kim, J.M., Porter, A., 2002. A history-based test prioritization technique for regression testing in resource constrained

environments. In: Proceedings of the International Conference on Software Engineering (ICSE 2002), Orlando, FL, USA, pp.119–129.

[9] Park, H., Ryu, H., Baik, J., 2008. Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing. In: Proceedings of the 2nd International Conference on Secure System Integration and Reliability Improvement (SSIRI 2008), Yokohama, Japan, pp. 39–46.

[10] Walcott K R, Soffa M L, Kapfhammer G M, Roos R S. Time-aware test suite prioritization. In: Proceedings of the 2006 ACM International Symposium on Software Testing and Analysis. 2006, 1–12

[11] Arafeen M J, Do H. Test case prioritization using requirements based clustering. In: Proceeding of the 6th IEEE International Conference on Software Testing, Verification and Validation. 2013, 312–321

[12] Zainab Sultan, Rabiya Abbas, Shahid Nazir, S.Asim. "Analytical Review on Test Cases Prioritization Techniques: An Empirical Study", International Journal of Advanced Computer Science and Applications, 2017.

[13] Miso Yoan,Eunyoung Lee,Mikyoung song and Byoungju choi, A Test case Prioritization through Correlation of Requirement and Risk,Journal of software engineering and applications,2012,pp.823-835

[14] Hsu, Yen-Ching, Kuan-Li Peng, and Chin-Yu Huang. "A study of applying severity-weighted greedy algorithm to software test case prioritization during testing", 2014 IEEE International Conference on Industrial Engineering and Engineering Management, 2014.

[15] Musa, Samaila, Abu-Bakar Md Sultan, Abdul- Azim Bin Abd-Ghani, and Salmi Baharom."Software Regression Test Case Prioritization for Object-Oriented Programs using Genetic Algorithm with Reduced-Fitness Severity", Indian Journal of Science and Technology,2015.

[16] Saha, R. K, Zhang, L., Khurshid, S., and Perry, D. E. 2015.An Information Retrieval Approach for Regression Test Prioritization Based on Program Changes. *In Proceedings of the 37th International Conference on Software Engineering (ICSE'15).* 268-279.

[17] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," JSTVR, pp. 67–120,Mar. 2010.

[18] N. Schneidewind and H.-M. Hoffman, "An experiment in software error data collection and analysis," IEEE TSE, vol. 5,no. 3, pp. 276–286, May 1979.

[19] W. Zage and D. Zage, "Evaluating design metrics on large scale software," IEEE TSE, vol. 10, pp. 75–81, 1993.

[20] T. Graves, "Predicting fault incidence using software change history," IEEE TSE, vol. 26, pp. 653–661, Jul. 2000.

[21] P. Nagahawatte and H. Do, "The effectiveness of regression testing techniques in reducing the occurrence of residual defects," in ICST, Apr. 2010, pp. 79–88.