Test Case Prioritization with Textual Comparison Metrics

Rooster Tumeng, Dayang Norhayati Abang Jawawi, Mohd Adham Isa
Department of Software Engineering, Faculty of Computing
Universiti Teknologi Malaysia
Skudai, Johor, Malaysia
rooster2@live.utm.my, dayang@utm.my, mohdadham@utm.my

***Abstract*—Regression testing of a large test pool consistently needs a prioritization technique that caters requirements changes. Conventional prioritization techniques cover only the methods to find the ideal ordering of test cases neglecting requirement changes. In this paper, we propose string dissimilarity-based priority assignment to test cases through the combination of classical and non-classical textual comparison metrics and elaborate a prioritization algorithm considering requirement changes. The proposed technique is suitable to be used as a preliminary testing when the information of the entire program is not in possession. We performed evaluation on random permutations and three textual comparison metrics and concluded the findings of the experiment.**

Keywords—textual comparison, test case prioritization, regression testing

I. INTRODUCTION

Regression testing is a testing action that is performed when changes are performed on a current programming software implementation. Performing such a test is not cheap as the cost and time involved in testing typical software used in industries involve thousands of test cases. Test case prioritization [1] searches for the best ordering of test cases for testing, so that any objectives specified prior to the regression test would be achieved. For instance, such example is to maximize the number defects detected the earliest.

Requirements to test case changes [2] has a significant lack of research carried out in communicating them. In this context, a change performed in requirement elicited for a software is often not reflected in the test case. Conventional prioritization techniques cover only the methods to find the ideal ordering of test cases. Indeed, the current prioritization technique practice is to assign a priority to each test case based on a certain criteria. However, a change in the requirement is not traceable in the test case. This significant weakness might render the set of prioritized test cases to be useless as new requirements emerge. In textual comparison, based on earlier work carried out by [3], is a method of assigning a higher priority to a test case that is the most different in terms of its text count from the compared test case next to it. The most different test case has a high count of textual difference will be assigned a higher priority through a prioritization algorithm.

In this paper, we propose a technique which prioritizes test cases accounting for requirements changes traceable to the test cases. This technique is suitable to be performed as a preliminary testing of a software when the information of coverage is expensive to be gathered. The idea of textual comparison of test cases is that it depends only on the strings with no attention paid to the information of the software under test. In this proposed technique, a test case is treated as a string of characters. The difference of these string of characters will then be assumed as priority. As new requirements emerge, a new priority will be assigned to the requirements to the best of the requirement engineer's knowledge. This requirement priority will then be reflected to the prioritized test cases. Different approaches to perform textual comparison is performed using both classical and non-classical textual comparison metrics. The combination of textual comparison metrics in prioritization of text cases have not been explored yet by other researchers in this field.

Textual comparison of characters of strings in a test case provides a number of benefits when used in test case prioritization. Firstly, the difficulty in gathering the code coverage information is ignored as the textual difference does not depend on such information. Secondly the approach produces the best test cases sequence from a series of test cases provided in each test suite making the effort as minimal as possible in detecting defects earlier.

The rest of the paper is organized as the following. Section II presents the classical and non-classical textual dissimilarity metrics and their comparison. Section III presents the prioritization algorithm utilizing the textual comparison metrics to prioritize test cases on a small example. Section IV presents the evaluation criteria used in this experiment. Section V discusses the findings of the experiment including random permutation and the classical and non-classical string distances through Siemens Test Suite bench mark programs. Section VI lays out the conclusions of this work.

II. TEXTUAL COMPARISON METRICS

A. Manhattan Textual Comparison

Manhattan textual comparison [3] is a measure of distance of two set of strings based on their ASCII code (or numbering code). A string with n size is viewed in n-dimensional planar

978-1-4673-8227-4/15/$31.00 ©2015 IEEE

as characters of vector. A classical textual comparison for Manhattan can be calculated with the following Equation (1):

$$d_{man} = \sum_{i=1}^{n} |x_i - y_i| \quad (1)$$

A shorter string, between "abc 1" and "abc.xyz -1" is filled with `char (0)`. For instance, "abc 1**" where "*" stands for `char (0)`. The measure can be calculated for the following textual difference between "abc 1" and "abc.xyz -1" as [1]:

|97-97|+|98-98|+|99-99|+|32-46|+|49-120|+
|0-121|+|0-122|+|0-32|+|0-45|+|0-49|
=454

This measure of textual distance shows that the distance of the ACII numerical unit is 454.

B. Jaro-Winkler Textual Comparison

Jaro-Winkler textual comparison [4] is a measure of similarity between two set of strings. It is the extension of the earlier developed textual comparison known as Jaro for duplicate detection. Jaro-Winkler is the improvement of the measure of similarity between two strings assuming that fewer errors are found at the beginning of names. It thus has a higher similarity measure whereby a higher similarity will have a value approaching 1 while absolute no similarity will have a value of 0. Jaro-Winkler textual comparison can be computed through calculating the Jaro distance in (2) and followed by the Winkler extension in (3) as follows:

$$d_{jaro}(s_1, s_2) = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) \quad (2)$$

$$d_{jaro-winkler} = d_{jaro}(s_1, s_2) + \frac{s}{10}(1.0 - d_{jaro}(s_1, s_2)) \quad (3)$$

For instance, given s_1 = abc 1, s_2 = abc.xyz -1, its m, maximum number of similar string is 3, number of string $|s_1|$ = 5, $|s_2|$=10 and transposition t =0 as 'a', 'b', and 'c', are in the same order in s_1 and s_2. The $d_{jaro}(s_1, s_2)$ is 0.733333. The $d_{jaro-winkler}$, hence, is 0.81333. The dissimilarity score for Jaro-Winkler:

$$JW_{dissimilarity} = 1 - d_{jaro-winkler}$$

$$JW_{dissimilarity} = 1\text{-}0.81333 = 0.18667$$

The result shows that the two strings are only 18.667% different from each other.

C. Jaccard Textual Comparison

Jaccard textual comparison [5] is a measure of similarity of two set of samples. The samples in this context can be test cases. Jaccard textual comparison can be measured by using Equation (4):

$$Jac(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (4)$$

[1] ASCII codes for a, b, c, white space, -, ., x, y, z, 1 are 97, 98, 99, 32, 45, 46, 120, 121, 122 and 49.

The calculation of Jaccard distance can be performed on the earlier examples of test cases, s_1 = abc 1, s_2 = abc.xyz -1. The measure of similarity for Jaccard distance between s_1 and s_2 is as the following:

$$Jac(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|} = \frac{|a, b, c|}{|a, b.c, ., x, y, z, 1, -1|} = \frac{3}{9} = 0.33333$$

The Jaccard index for s_1 and s_2, thus is 33.333%. Jaccard textual comparison metric can be applied to a variety string lengths. Jaccard dissimilarity score can be defined as following:

$$Jac_{dissimilarity} = 1 - Jac(s_1, s_2)$$

$$Jac_{dissimilarity} = 1 - 0.33333 = 0.66667$$

The Jaccard dissimilarity score for s_1 and s_2 is 66.667%

III. Prioritization Technique

A. Test Suite Prioritization Strategy

Researchers in [3] introduce a classical string distance-based prioritization strategy. The goal of the prioritization strategy is to find the best ordering of test cases yielding a higher detected faults. This approach shows encouraging findings motivating us to explore further on textual comparison metrics to prioritize test cases. Our work proposes a combination of classical and non-classical textual comparison scores aims to yield a higher detected faults while satisfying changing requirements. In order to apply the prioritization technique, the following inputs are needed:

- Test suite containing test cases to be prioritized;
- Set of changing requirements;
- Classical and non-classical textual comparison metrics scores measured by their performance function;
- Prioritization algorithm.

B. Prioritization Definition

The researchers in [6], define the test case prioritization problem as follows:

Given:

T, a test suite;

PT, the set of permutations of T, and

f, a performance function from PT to the real numbers

Find:

$T' \in PT$ such that $(\forall T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$

PT, is the set of all permutations of T and thus the goal of the prioritization is to satisfy the performance function, f . The problem definition of test case prioritization remains the same except for the fitness function that is often defined depending on the prioritization problem and test data.

C. Requirements to Test Case Traceability

According to researchers in [7], requirements traceability is described as the capability to define and trace the lifetime of a requirement, in straightforward and reverse course. As shown in Fig. 1, requirements, code and test cases are interrelated which must be traceable in our testing effort such as the assertion of weight priority in test cases.

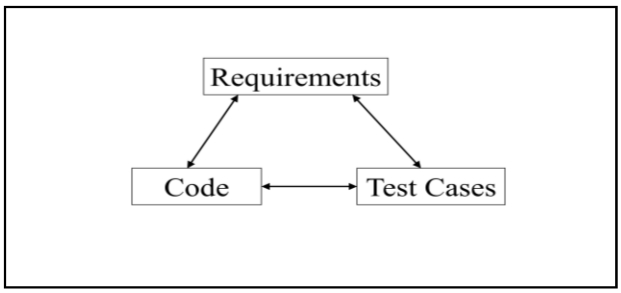

Fig. 1. Requirements to Test Cases Traceability

The assignment of requirements priority in this work is based on the knowledge of the tester. Requirement priority is assigned on a 0.1 to 1 scale. The highest priority is 1. A requirement traceability matrix is established according to the following Table I. In this table, we will utilize 7 sample test cases as used by researchers in [3] as shown in Table II. The following sample of values for the requirements are assigned randomly to demonstrate the changing requirements priority as shown in Table I.

TABLE I. REQUIREMENTS TO TEST CASE PRIORITY ASSIGNMENT

Test Case	Manhattan-Jaro-Winkler Score						
	TC1	*TC2*	*TC3*	*TC4*	*TC5*	*TC6*	*TC7*
TC1	1					0.4	
TC2	1		0.8				
TC3		0.9					
TC4				0.6		0.4	
TC5		0.9			0.5		
TC6	1			0.6			0.1
TC7		0.9					0.1

D. Performance Function and Prioritization Idea

The idea of the prioritization is to use a combination of classical and non-classical textual comparison metrics to determine the dissimilarity score. The highest dissimilarity score will have the highest priority. The dissimilarity score between a test case, t_i and a subsequent test case, $\{t_1,...t_n\}$ in test suite, *T,* for Jaro-Winkler and Jaccard metrics only is shown in Equation (5), while Equation (6) is only intended to be used by Manhattan metric.

$$ds : [1 - SimilarityScore(t_i, \{t_1,...t_n\})] \times R_{priority} \quad (5)$$

$$ds : [DissimilarityScore(t_i, \{t_1,...t_n\})] \times R_{priority} \quad (6)$$

The priority assignment for requirements, $R_{priority}$ is as follows.

$R_{priority}$ = Assigned through knowledge of tester

The corresponding performance function, in this work is generalized in Equation (7) utilizing the textual dissimilarity score. The performance function rewards the highest dissimilarity score as the most far apart test case in the test suite, *T*.

$$f(P) = \sum_{i=1}^{n} ds(t_i, \{t_1,...,t_n\}) \quad (7)$$

This fitness function chooses the most dissimilar test case score from the set of already prioritized test case with consideration on requirements changes. The test cases are ordered by using 2-optimal algorithm. 2-optimal algorithm at each iteration picks a pair of test cases with the highest dissimilarity score combined and place them at the top of the set of prioritized test cases.

TABLE II. CAESAR CIPHER TEST CASES

Test Case	Caesar Cipher		
	Input	*Move*	*Output*
TC1	abc	1	bcd
TC2	abc.xyz	-1	zab.wxy
TC3	a.Z	27	b.A
TC4	AaZz	0	AaZz
TC5	xyz	1	yza
TC6	é é é	4	é é é
TC7	a..z:A..Z	1234567	j..i:J..I

E. Dissimilarity Score of a Test Case

The dissimilarity score of a test case helps in the determining the position in the prioritized test suite. By treating the test case based on its dissimilarity score as a two-planar vector, we can plot the dissimilarity score. The prioritization process will take two test cases with the highest fitness function combined. As shown in Table III, the multiplication of the respective dissimilarity score and requirement priority yield the value in each respective column.

Based on Table III, the dissimilarity score for a hybridized dissimilarity score namely Jaro-Winkler-Jaccard Score is based on the average score of the combined two textual comparison metrics. Each of the score is based on the calculation of non-classical textual metric dissimilarity that is further multiplied with $R_{priority}$ resulting in a complete dissimilarity score ds .

TABLE III. DISSIMILARITY SCORE OF CAESAR CIPHER TEST CASES

Test Case	Manhattan Score						
	TC1	*TC2*	*TC3*	*TC4*	*TC5*	*TC6*	*TC7*
TC1	997	454	117	197	69	399	847
TC2	997	0	798	527	523	814	815
TC3	117	459	0	198	186	359	730
TC4	197	527	198	598	266	399	748
TC5	69	897	186	266	499	435	916
TC6	997	814	359	598	435	0	100
TC7	847	897	730	748	916	997	100
Test Case	**Jaro-Winkler Score**						
	TC1	*TC2*	*TC3*	*TC4*	*TC5*	*TC6*	*TC7*
TC1	1	0.2	0.4	0.4	0.4	0.4	0.4
TC2	1	0	0.8	0.4	0.3	0.6	0.4
TC3	0.4	0.9	0	0.3	0.6	0.6	0.3
TC4	0.4	0.4	0.3	0.6	0.4	0.4	0.4
TC5	0.4	0.9	0.6	0.4	0.5	0.5	0.4
TC6	1	0.6	0.5	0.6	0.5	0.1	0.1
TC7	0.4	0.9	0.2	0.4	0.4	0.5	0.1
Test Case	**Jaccard Score**						
	TC1	*TC2*	*TC3*	*TC4*	*TC5*	*TC6*	*TC7*
TC1	1	0.6	0.9	0.9	0.9	0.4	0.9
TC2	1	0	0.8	0.8	0.6	1	0.8
TC3	0.9	0.9	0	0.7	1.0	1.0	0.6
TC4	0.9	0.8	0.7	0.6	0.9	0.4	0.7
TC5	0.9	0.9	1	0.9	0.5	1.0	0.9
TC6	1	1.0	1.0	0.6	1	0	0.1
TC7	0.9	0.9	0.6	0.7	0.9	0.9	0.1
Test Case	**Jaro Winkler-Jaccard Score**						
	TC1	*TC2*	*TC3*	*TC4*	*TC5*	*TC6*	*TC7*
TC1	1	0.4	0.7	0.7	0.7	0.4	0.7
TC2	1	0	0.8	0.6	0.5	0.8	0.6
TC3	0.7	0.9	0	0.5	0.8	0.8	0.5
TC4	0.7	0.6	0.5	0.6	0.7	0.4	0.6
TC5	0.7	0.9	0.8	0.7	0.5	0.8	0.7
TC6	1	0.8	0.8	0.6	0.8	0.1	0.1
TC7	0.7	0.9	0.4	0.6	0.7	0.7	0.1

F. 2-Optimal Prioritization Algorithm

Based on the work carried out by researchers in [9], greedy algorithm is concluded to perform sub-optimally in yielding the most optimum set of prioritized test cases and noted that 2-optimal algorithm yields a significantly better Average of Fault Detected measure. Hence, in this work, 2-optimal prioritization algorithm is applied to the proposed work.

Based on this 2-optimal algorithm, two test cases with the highest dissimilarity score are selected at every iteration. The prioritization order O, of a test suite T, is shown in Fig. 2. The algorithm is used in the ordering of the test suite implemented in Java for each dissimilarity score metric of Sect. II.

1. Calculate the dissimilarity score ds for every pair of test cases in T
2. Find two test cases $(t_{Highest}, t_{2ndHighest}) \in T$ with maximum ds combined, $T := T \setminus \{t_{Highest}, t_{2ndHighest}\}$, $O := \{t_{Highest}, t_{2ndHighest}\}$
3. While T is not empty
3.1 Find two test cases $(t_{Highest}, t_{2ndHighest}) \in T$ with maximum ds combined, $T := T \setminus \{t_{Highest}, t_{2ndHighest}\}$, $O := O.\{t_{Highest}, t_{2ndHighest}\}$ ($t_{Highest}$ and $t_{2ndHighest}$ are appended to the prioritization order)
4. Return P

Fig. 2. Proposed 2-optimal Algorithm as Prioritization Algorithm.

TABLE IV. SIEMENS TEST SUITE

Program	Detail		
	LOC	*Mutants*	*Number of Test Cases*
printTokens	726	7	4130
printTokens2	570	10	4115
replace	564	31	5542
schedule	412	9	2650
schedule2	374	9	2710
tcas	173	41	1608
totinfo	565	23	1052

IV. EVALUATION CRITERIA

In this proposed work, Average Percentage of Faults Detected (APFD) will be used to evaluate the proposed work. APFD [8] is a classical measurement metric to measure the percentage of faults that can be detected. A higher percentage signifies a better technique in detecting faults and is more desired. Equation (8) elaborates AFPD.

$$APFD = 1 - \frac{TF_1 - TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n} \quad (8)$$

n is the number of test cases, and m is the number of faults contained in program under test, P and TF_i is the index of the first test in T that exposes fault i. For instance, supposed we have 5 test cases, TC1, TC2, TC3, and TC4, and TC5 and supposed the faults are at TC4 and TC5. For a non-prioritized test suite ($TC1_1$, $TC2_2$, $TC3_3$, $TC4_4$, $TC5_5$), the AFPD calculation would be as follows,

$$APFD = 1 - \frac{4+5}{5*2} + \frac{1}{2*5} = 0.2$$

Meanwhile for a prioritized test suite ($TC5_1$, $TC4_2$, $TC2_3$, $TC3_4$, $TC1_5$), the AFPD would be as follows,

$$APFD = 1 - \frac{2+1}{5*2} + \frac{1}{2*5} = 0.8$$

A higher AFPD indicates that the test suite is more effective in detecting faults earlier.

V. Findings and Discussion

Based on the proposed work in this paper, the individual result of every dissimilarity score metric might produce slightly different result. Hence, we are interested in answering these questions:

1) On comparison with random ordering, is the prioritized set of test cases more superior in discovering faults?
2) Which of the textual comparison metrics produce a promising result?

The experiment was carefully conducted on all seven Siemens Test Programs. 20 carefully selected test cases from each of the test pools were hand-picked to allow a balanced number of test cases that could detect faults and test cases that could not detect faults. Each of the test case selected test suite were rearranged according to Manhattan, Jaro-Winkler, Jaccard and Jaro-Winkler-Jaccard metrics. Upon completion of the rearrangement of the test cases order, the test cases were prioritized based on our proposed algorithm, 2-optimal algorithm.

In order to measure the performance of these dissimilarity measure-based prioritization, we performed the calculation of Average Fault Percentage Detected for all of the test cases. In average, the AFPD of the 7 Siemens Test Suite programs were averaging 75% with having 40% as the lowest AFPD measure. Fig. 3 shows the result of AFPD for one of the programs, *printTokens.* The AFPD results of this *printTokens* program is consistent with the trend of the remaining 6 Siemens Test Suite Programs. The dissimilarity metrics that show promising results are Jaccard metric and the combination of Jaro-Winkler-Jaccard metric. This result shows that these two metrics are superior to Manhattan-based metric.

We also recorded the randomly prioritized test suites as the worst performing test suite throughout the 7 programs. These experimental results show that the carefully prioritized test suite coupled with our proposed 2-optimal algorithm performs more efficient in terms of AFPD measures. It is also noted that throughout the experiment, AFPD averages higher for prioritized suites with string metrics. Since this work emphasizes on changing requirements, we simulated an enforced priority insertion to certain test cases in one program, *printTokens.* We discovered that Jaro-Winkler ends up with too many similar distances making it less likely to be intuitive in the prioritization of the test suite.

Jaccard metric shows to be a more suitable candidate to be used in re-ordering the test cases with an enforced simulated priorities of changing requirements. Jaccard metric seems to be more deterministic in the re-ordering of test cases. Deterministic dissimilarity metric as seen in this experiment yields a higher AFPD measure.

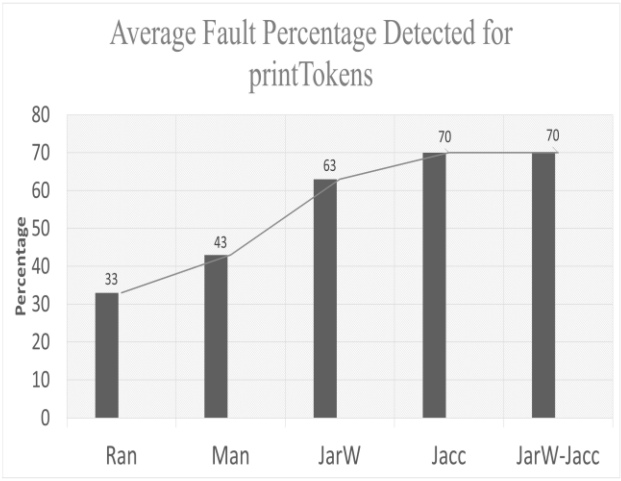

Fig. 3. Average Faults Percentage Detected (AFPD) for *printTokens.* (Ran-Random, Man-Manhattan, JarW-Jaro-Winkler, Jacc-Jaccard, JarW-Jacc-Jaro-Winkler-Jaccard)

Higher AFPD measures assume that faults are discovered earlier in testing. Since Jaccard metric is capable of yielding AFPD, more in-depth work is needed to be carried out in the nature of changing requirements involving new priorities order. Upon the insertion of new priorities the AFPD measure averages around 30%-60% throughout the 7 test suites with Jaccard metric averages higher throughout the 7 test suite programs. However it should be noted that AFPD is less favorable in the context of enforced new priority order simulating changing requirements importance as the priority and attention is given to test cases that are required to be executed first in a software program which are deemed as more important.

Random prioritization shows the least AFPD in this experiment. AFPD measure enables a tester to tell whether the prioritized test suite is capable in finding defects as early as possible. Inducing requirement priorities to simulate the changing of requirements in this experiment is limited to the knowledge of the tester. However, in this paper, through the combination of dissimilarity string-based approaches and a

prioritization algorithm, we were able to prioritize the controlled experiments based on 7 Siemens Test Suite programs without biasness from the tester's judgement.

The performance of Jaccard metric surpasses other dissimilarity approaches compared in this work. It is a good indicator towards a more effective permutation technique whereby the test cases are assigned with a certain dissimilarity based weight before being prioritized with a prioritization algorithm, which in the case of this research is a 2-optimal algorithm.

As a summary, Jaccard metric is seen as a more suitable candidate in performing prioritization coupled with a prioritization algorithm as it is more effective in discovering faults earlier.

VI. Conclusion

In this work, a new set of prioritization approach based on non-classical string dissimilarity measure coupled with a prioritization algorithm is proposed. The prioritization technique also caters to requirement changes that are enforced into affected test cases through enforced priority weight. The proposed technique is a suitable technique to be used as a preliminary testing when the information of the entire program is not in possession. The idea of this preliminary testing is beneficial in developing a software where early detection of faults would cost less to be fixed during a software development life cycle.

We developed a technique that utilizes both one classical and two non-classical string dissimilarity-based measure in order to assign distance-based weight to each test case. This allows an unbiased assignment of priority weight to a test case. This is especially useful when working with a large pool of test cases whereby a quick and efficient priority weight is assigned to every test case that considers the whole test suite. Manhattan distance which is a classical string dissimilarity measure is utilized in this work, while two non-classical string dissimilarity, Jaccard and Jaro-Winkler dissimilarity were used as a mean of comparison to determine which technique of priority weight assignment would yield a better result in the testing performed. The result that was sought after in this paper is the Average Faults Percentage Detected (AFPD) measure. The AFPD measure was performed after the prioritization of the test cases with the assigned string dissimilarity-based weight through 2-optimal algorithm.

The findings of the experiment confirms that Jaccard metric is more efficient as it is capable to yield a higher AFPD measure compared to other string dissimilarity-based metrics.

Acknowledgment

The work is fully funded by Fundamental Research Grant Scheme, vote number 4F303 and Research University Grant Scheme, vote number 05H75 under the Ministry of Education, Malaysia as well as Science Fund, vote number 4S064 under the Ministry of Science, Technology and Innovation, Malaysia. We would like to extend our gratitude to these ministries and Universiti Teknologi Malaysia for their financial support. We would also like to thank the members of Embedded & Real-Time Software Engineering Laboratory (EReTSEL) for their feedback and continuous support.

References

[1] S. Elbaum, A. Malishevsky and G. Rothermel, "Prioritizing test cases for regression testing," SIGSOFT Softw. Eng. Notes, vol. 25, no. 5, pp. 102-112, 2000.

[2] M. Unterkalmsteiner, R. Feldt and T. Gorschek, "A taxonomy for requirements engineering and software test alignment," ACM Transactions on Software Engineering and Methodology, vol. 23, no. 2, pp. 1-38, 2014.

[3] Y. Ledru, A. Petrenko, S. Boroday and N. Mandran, "Prioritizing test cases with string distances," Autom Softw Eng, vol. 19, no. 1, pp. 65-95, 2011.

[4] W. E. Winkler, "String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage," in Proceedings of the Section on Survey Research, 1990, pp. 354-359.

[5] W. Cohen, P. Ravikumar, and S. Fienberg, "A comparison of string metrics for matching names and records," in Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 2003, pp. 73-78.

[6] G. Rothermel, R. H. Untch, C. Chengyun, and M. J. Harrold, "Prioritizing test cases for regression testing," IEEE Transactions on Software Engineering, vol. 27, pp. 929-948, 2001.

[7] O. C. Gotel and A. C. Finkelstein, "An analysis of the requirements traceability problem," in Proceedings of the First International Conference on Requirements Engineering, 1994, Colorado, USA, 1994, pp. 94-101.

[8] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: a family of empirical studies," Software Engineering, IEEE Transactions on, vol. 28, pp. 159-182, 2002.

[9] L. Zheng, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," IEEE Transactions on Software Engineering, vol. 33, pp. 225-237, 2007.