RDCC: An Effective Test Case Prioritization Framework using Software Requirements, Design and Source Code Collaboration

Md. Saeed Siddik
Institute of Information Technology
University of Dhaka, Bangladesh
siddik.saeed@gmail.com

Kazi Sakib
Institute of Information Technology
University of Dhaka, Bangladesh
sakib@iit.du.ac.bd

***Abstract*—Test case prioritization is a technique for selecting those test cases, which are expected to outperform for determining faulty modules earlier. Different phases of software development lifecycle represent the total software from different point of views, where priority module may vary from phase to phase. However, information from different phases of software development lifecycle is rarely introduced and no one integrates that information to prioritize test cases. This paper presents an effective test case prioritization framework, which takes software requirements specification, design diagrams, source codes and test cases as input and provides a prioritized order of test cases using their collaborative information as output. Requirement IDs are split into words or terms excluding stop words to calculate requirements relativity. Design diagrams are extracted as readable XML format to calculate the degree of interconnectivity among the activities. Source codes are parsed as call graphs where vertices and edges represent classes, and calls between two classes respectively. Requirements relativity, design interconnectivity and class dependencies are multiplied by their assigned weight to calculate final weight and select test cases by mapping the customers' requirements and test cases using that weight. The proposed framework is validated with an academic project and the results show that use of collaborative information during prioritization process can be beneficial.**

Keywords. Software Engineering, Software Testing, Test Case Prioritization, UML and Code Collaboration

I. Introduction

Test Case Prioritization (TCP) re-orders the test cases for providing earlier feedback to software testers and managers about faulty modules [1], [2]. TCP may reduce the cost and time in testing phase by working on critical sections earlier. Testing is a vital phase of Software Development Life Cycle (SDLC) ensuring the correctness and quality of final products. It is a process for evaluating software by detecting differences between given input and expected output. However it costs more than 40% of total development budget and time [1], [3]. Although TCP can minimize the testing time and cost, without executing the whole test suites, test case prioritization is really difficult to achieve.

A representation of information from all phases of SDLC (such as requirements, design and code) has some uniqueness like use-case, UML diagram, design pattern, etc [4], [5]. Those phases can individually represent a software using their own point of views, which can be effective for TCP. Software Requirements Specification (SRS) documents are prepared by requirements engineers using direct interaction with customers or end users. On the other hand, software or system designers prepare design documents and diagrams on the basis of that SRS. Finally source codes are developed by software developers based on those design diagrams. These three phases represent the total software from different viewpoints with various priorities. That is why collaborative information from every phase is important to make an effective TCP approach.

Several test suite optimization approaches have been proposed for improving the efficiency of testing like [6], [7], [8]. Rothermel et al. [1] and Elbaum et al. [9] presented techniques to prioritize test cases based on total statement, branch and function coverage etc. Rothermel et al. analyzed nine distinct approaches for TCP [1]. Arafeen et al. presented TCP technique using requirements based clustering that incorporates traditional code analysis information for improving TCP techniques [10]. However their work failed to incorporate user defined prioritization and static code information. Srivastva et al. presented TCP techniques based on requirements and risk factors [11]. Their work integrates priority factors from customers, developers and managers to reorder the test cases. However those prioritization is incomplete for not considering the static or dynamic source code information of software.

This paper proposes a software Requirements, Design and Code Collaboration (RDCC) framework for TCP. This framework consists of three layers where the top layer takes SRS, design diagrams, source codes, and test suites as input. The middle and main layer is RDCC service layer includes the processing units of SRS, diagrams and source codes. Requirements are considered as a pool of words and extracted by excluding the stop words. Design diagrams like UML are extracted as readable XML format, and the connectivity among the major activities are calculated. An underlying dependency call graphs are generated based on the static source code analysis where the vertices and edges represent the classes and calls between two classes. Weights from the design documents and source codes are assigned by using design connectivity and class dependency. The final weighted value is calculated using the sum of total weights from different phases which are the output of the middle layer of the framework. The bottom layer maps test cases with customer requirements using the weighted values from middle layer. The final output of this framework is an ordered list of test cases for execution.

978-1-4799-6288-4/14/$31.00 ©2014 IEEE

A case study is performed for the proposed framework on an academic project developed and managed by students, titled as "Academic Time Synchronization Project (ATSP)". This project has 10 distinct test cases for 7 requirements. The top layer of this framework generates IDs from the requirements, design diagrams and source codes of this project. The middle layer processes those IDs to calculate prioritized values. The final calculated values of seven $RDCCID$s are $0.4785, 0.4158, 0.396, 0.4455, 0.3993, 0.4653,$ and 0.3696 respectively. The new order of Test Cases (TC) is $1, 4, 8, 10, 3, 2, 7, 6, 9,$ and 7, which is the final output of the framework. After executing test cases according to that order, results show that collaborative information approach can improve the effectiveness of TCP.

II. Literature Review

TCP is a technique to find the ideal ordering of test cases for software testing to obtain maximum benefits for software testers. Rothermel et al. are one of the pioneers of software TCP research who proposed and evaluated the TCP approaches [1]. They described several techniques to prioritize test cases like random approaches, rate of coverage base approaches etc. They reported an empirical study conducting with seven C programs containing 1000 to 5500 test cases to study the effectiveness of different prioritization techniques. Their process has been implemented based on coverage based prioritization approach. However there is a sizable performance gap between prioritization heuristics and optimal prioritization, which are not being bridged by their proposed technique.

Srikanth et al. presented TCP technique for system level testing, based on requirements specification to extend the code-coverage prioritization techniques for both new and regression tests [12]. Their work introduced four distinct factors for prioritizing user requirements named, customer assigned priority, requirements volatility, implementation complexity, and fault proneness. To determine the effectiveness of their technique, they conducted two phased feasibility study in four similar student projects. Although those approaches can incorporate customer requirements priority with the test suite, there is no direction for relating requirements to the design and implementation modules.

A multifaceted prioritization strategy has been explored with two prioritization facts one is priority from customers, developers, and managers, and another is severity of risk factors that occurs in requirements [11]. They assigned different values to requirements from customers and developers which has been added with the risk factor of those requirements. Probability factors from 1 to 10 have been assigned to each requirement, based on their importance. This process finally delivered a numeric value of weighted risk and requirements priority. However they failed to consider the static or dynamic source code information of software.

Srivastava et al. presented Echelon, a TCP framework that runs under a Windows environment [13]. Test cases are prioritized based on the changes made to a program. Echelon uses a binary matching system that can accurately compute the differences at the basic block level between two versions of the program in binary form. Test cases are ordered to maximize coverage of affected regions using a fast, simple and intuitive heuristic. Echelon has been found to be quite effective on large binaries. It would be a better one, if this framework will compatible with all operating platforms.

Hyunsook Do et al. reported an empirical study to assess the effects of time constraints on the cost and benefits of prioritization techniques [14]. Three different experiments have been presented for determining cost effective regression testing. The assessment was made for five java programs ranging from 7600 to 80400 lines of codes, 78 to 912 test cases. The empirical results show that the time constraints played a significant role in determining the cost effectiveness of the prioritization techniques.

A graph-theoretic framework for test case prioritization named PHALANX has been presented by addressing the limitations of implementation complexity in testing process [15]. It abstracted test cases into a test-case dissimilarity weighted graph, where nodes specify test cases and weighted edges specify user defined proximity measures between test cases. Edge weight has been calculated using the edit distance [16] between two test cases. Two mechanisms named Fiedler (spectral) ordering and a greedy approach have been explored for finding the maximum weighted path. Experimental results noticed that this framework can detect major faults after executing the first 20% of the ordered test cases.

Dynamic prioritization is an approach which involves the changing order of test cases during the software testing phase [17]. It presents a framework for mitigating the challenges in dynamic industrial code changing. Various types of challenges have been introduced in their work such as environment and resource related challenges etc. An example scenario has also been demonstrated for dynamic prioritization where the order of test suite has been changed automatically. They ignored those test cases which are being obsolete. However the test cases which are already being obsolete may be important after next changes in regression testing.

Arafeen et al. presented TCP for regression testing using requirements-based clustering [10]. They investigated the clustering approach that incorporates traditional code analysis information for improving TCP techniques. Their approach consisted of five main activities named as requirements clustering, requirements tests mapping, prioritization of test cases for each cluster, cluster prioritization, and test case selection from the clusters. They reported the experimental results which have been implemented on two in-house developed software named iTrust and Capstone containing 42 and 142 test cases respectively. However their work failed to incorporate user defined prioritization and static code information.

The review of the existing literature has shown that various prioritization techniques have been proposed for regression testing like branch coverage based approach, probabilistic approach etc. Very few researchers addressed the software requirements information for TCP which are incomplete because, none of the work directly proposed any method that incorporates requirements, designs and source codes together.

III. RDCC Framework

This section describes the proposed framework including the architecture and activities. RDCC is a framework for TCP

978-1-4799-6288-4/14/$31.00 ©2014 IEEE

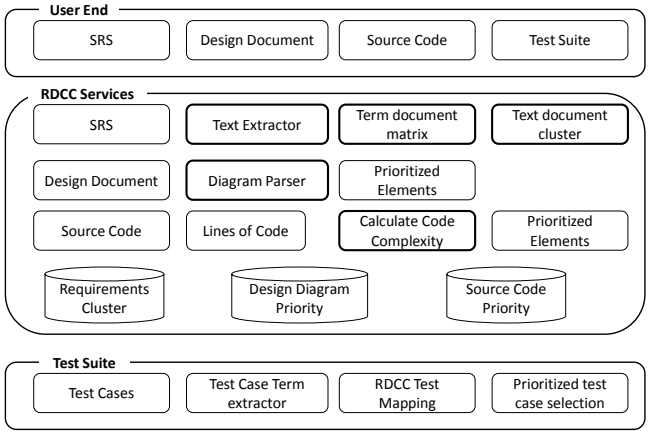

Fig. 1. Architectural component stack of RDCC

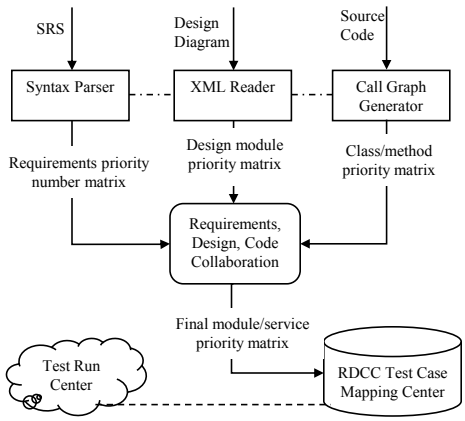

Fig. 2. Internal interaction of RDCC activities

where information from every phase of SDLC is combined to make a decision for test cases ordering. In SDLC, SRS documents are equipped by requirement engineers using direct interaction with customers or end users. Software designers design documents and diagrams for the development phase on the basis of that SRS documents. Finally source codes have been developed by software developers based on those design diagrams.

These three phases represent the total software from different point of views with distinct priority. The proposed framework collaborates information from all phases of SDLC to identify efficient order of test cases. This framework takes SRS, design documents, source codes and test suites to provide an ordered list of test cases.

A. Architecture of RDCC Framework

The architectural component stack of the proposed framework is presented in Fig. 1, including various shapes like box, cylindrical shape etc. Boxes having thick border in Fig. 1 are the core elements of the framework, which focus on the overall major activities of RDCC. The boxes containing slim boarder are the supporting elements of the main activities. The storage of calculated values from different phases of SDLC are presented by cylindrical shape in Fig. 1. This framework is separated into three layers: User End, RDCC Service, and Test Suite Processing layer.

The top layer is composed of user end components because this layer takes direct input from customers. Those inputs (like SRS, source code etc.) are the processing elements of next RDCC service layer. The top level overview of this framework is consisted with four core sections of SDLC named (i) SRS, (ii) Design Documents, (iii) Source Code, and (iv) Test Suite. The second layer is the RDCC Service Layer which is the principle layer of this framework, where all the major activities are executed including the parsing of SRS, design documents and source codes. At the end of this layer, those parsing information are collaborated for TCP. The third and final layer is test suite processing layer where every test case is analyzed and mapped with prioritized RDCC module. Test cases are indexed based on previous RDCC information and ordered for executing in testing phase.

B. Activities of RDCC Framework

The proposed framework consists of six major activities named as requirements specification clustering and priority, design diagram priority, source code priority, RDCC module prioritization, RDCC test case mapping, and test case prioritization and selection. The internal interaction among the major activities including their action states have been presented in Fig. 2. The following subsection describes in details:

1) Software Requirements Specification Clustering and Priority : A list of well documented requirements is mandatory for this phase to parse the representative information. Textual similarities are implemented among those requirements for creating clusters. Textual similarity means grouping the same text in terms of syntactic and semantics. It has been studied in the field of data mining for clustering documents and information retrieval [18]. All the requirements create some clusters based on the similarity distribution of words. This process includes three tasks: text extraction, term document matrix, and term clustering.

a) Text Extraction: Each requirement is considered as a pool of words including meaningful and meaningless. In this phase, the requirements are split into words. The stop words that have no specific meaning or that are not related to principle goal like articles, preposition, conjunction, etc. have been eliminated. After eliminating those stop words, all distinct terms across the requirements are identified for next phase.

b) Term Document Matrix: The selected distinct terms obtained from the previous step are used to create a term document matrix. In this matrix the rows and columns correspond to the requirements and the distinct terms respectively. The matrix can also list the frequency of words in the corresponding requirements. The proposed framework uses term frequency-inverse document frequency (tf-idf) [18] approach to create term document matrix.

The term frequency, $tf(t, r)$, is the number of occurrences of a term, t, in a requirement r. The inverse document frequency, idf is calculated by taking the logarithm of the ratio for the total number of documents and the number of documents containing the term. The $tf * idf$ values of all the terms used in requirement R are used in the next step.

TABLE I. TERM DOCUMENT MATRIX (FOR SAMPLE DATA)

Req. ID	academy	admission	payment	... ...
Req# 1	0.301	0	0	... ...
Req# 2	0	0.145	0.11	... ...
Req# 3	0.191	0.14	0	... ...
Req# 4	0.189	0.3	0.31	... ...

TABLE I shows an example of the term-document matrix for five requirements. After performing termextraction, a set of distinct terms including $academy$, $admission$, $payment$, etc. are identified. The $tf * idf$ values of every term for each requirement are calculated. For instance, the $tf * idf$ value of term $academy$ for Req.# 1, 2, and 4 is 0.301, 0.191, and 0.189 respectively. That means Req# 2 is more similar to Req# 4 than Req# 1 in term of $academy$.

c) Term Clustering: There are many algorithms for clustering such as hierarchical clustering, k-means clustering etc. Among all, k-means clustering is suitable for document clustering [18] and, which is used to cluster distinct term documents. K-means clustering is used in the proposed framework. In this algorithm, for r number of requirements and t number of terms, the k-means technique assigns each requirement to one of the clusters to minimize the inter-cluster sum of squares shown in Equation (1).

$$Sum(k) = \sum_{i=0}^{r}\sum_{j=0}^{t}(x(i,j) - \overline{x(k,j)})^2 \quad (1)$$

Equation (1) presents the k-means cluster forming approach, where the $\overline{x(k,j)}$ sign denotes the mean variable of a cluster and k is the number of cluster.

2) Software Design Priority : Software design documents priority is also an important phase of this framework. To process the designs, information from different diagrams like UML, sequence diagrams etc. are extracted. The UML Reader takes UML diagrams provided by developers and forwards those to the XML Converter component, because program cannot take any input directly from the diagrams. This leads to the conversion of the diagrams to a program readable format (e.g.: XML). The XML Converter produces XML files that can present the major activities and their related sub actions. The interconnectivity values among the major activities are calculated and prioritized using their relationship.

3) Source Code Priority: Source codes are one of the major elements of software containing the real development phases. Developers need to follow the naming code convention for this framework. If developers failed to follow naming convention, that time static code parser cannot determine the expected phase or the implementation of required requirements. Basically two types of source code parsers are used, named as static and dynamic parser [18]. Static source code parser is used to retrieve information before executing the code. The code complexity metric value is calculated using three types of information obtaining from source codes.

a) Lines of Code (LOC): It measures the total number of lines in a class. RDCC considers only the executable code. It ignores the comments and the blank line inside the code.

b) Nested Block Depth (NBD): It measures the number of nested statements in a method or class.

c) Nodal Degree (ND): Call graph is developed from the source code where classes or methods and the connection between two classes or methods are presented as vertex and edge of a graph respectively. The degree of a vertex is called the nodal degree of a class or method.

Using these variables for each class, Source Code Metric (SCM) is calculated using the Equation (2). The values of LOC, NBD and ND are divided by their maximum values. Finally, those results are averaged by Equation (2), where i represent the code class ID.

$$SCM_i = \frac{\frac{LOC}{max(LOC)} + \frac{NBD}{max(NBD)} + \frac{ND}{max(ND)}}{3} \quad (2)$$

4) RDCC Module Integration and Prioritization: After calculating the priority of every term in requirements, designs and source codes phase, comparative priority are calculated. The requirement IDs are used as RDCC ID in this framework. The priority of each RDCC ID is calculated by the division of the term priority value and the sum of all priority. Equation (3) presents the calculation procedure of comparative priority, where i and j represent the RDCC ID and term ID number respectively.

$$RDCC_i = \frac{\sum_{j=1}^{t} TID_{ij}}{\sum_{i=1}^{r}\sum_{j=1}^{t} TID_{ij}} \quad (3)$$

Equation (4) explains the collaborative viewpoints by calculating the values of every RDCC ID. The values from requirements specifications, design diagrams, and source codes calculated from phases 1, 2, and 3 are multiplied by their weight constant α, β, and γ respectively. The weight constants may vary from 0 to 1 and the sum of all three constants must be equal to 1. The sum of all multiplied weighted values calculates the final weight. This calculation are executed until all the RDCC IDs are processed.

$$W_i = \alpha \times R + \beta \times D + \gamma \times C \quad \forall_{i=1,2,3,\ \ldots\ \ldots,\ t} \quad (4)$$

$$Subject\ to: \ 0 \le \alpha, \beta, \gamma \le 1 \ \ and \ \ \alpha + \beta + \gamma = 1$$

W_i= Final weight of RDCC ID$_i$
t = Number of RDCC terms
R = Requirements prioritization values (0...1)
α = Requirements prioritization weight
D = Design prioritization values (0...1)
β = Design prioritization weight
C = Code prioritization value (0...1)
γ = Code prioritization weight

5) RDCC Test Cases Mapping Resolution: After obtaining the values of RDCC, test cases traceability matrix has been generated to collect test cases that are associated to each requirement cluster. Fig. 3 summarizes the process where two

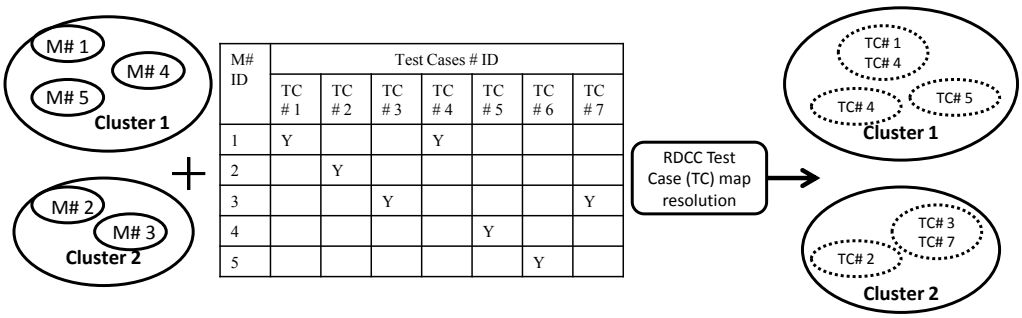

Fig. 3. RDCC Test Cases Mapping Resolution

ovals on the left side represent the clusters of requirements. In Fig. 3, cluster 1 contains requirements 1, 4 and 5. The Fig. represents the requirement-tests traceability matrix. Test Cases (TC) ID 1 and 4, associated to Req# 1. The RDCC-tests mapping resolution process obtains the clusters of test cases (the ovals on the right side of Fig. 3) by identifying the requirements and their corresponding test cases from the matrix. Cluster 1 on the right side of Fig. 3 contains four test cases (TC#1, TC#4, TC#5, and TC#6) that are associated to requirements 1, 4, and 5.

6) Test Cases Prioritization and Selection: This phase maps the test cases with RDCC items. The weighted priority sum of each RDCC# ID are calculated by adding the weighted values from different phases of SDLC. Finally, test cases are reordered decreasingly based on that sum of weighted values. If the weighted priority of requirements 1, 4 and 5 in Cluster 1 are $23.06, 19.14$, and 20.10 respectively, the total sum of Cluster 1 is 63.3. Using the same way the sum of Cluster 2 is 37.7. Finally the test cases order is:

- First phase: TC#1, TC# 4, TC#6, TC#5
- Second phase: TC#3, TC#7, TC#2

IV. Case Study

In order to demonstrate and validate the proposed framework, a case study is presented titled "Academic Time Synchronization Project (ATSP)", developed and managed by students. This project has 7 requirements and 10 test cases. It can synchronize 300 devices using Barkley Active time server algorithm. The requirements of this project are given bellow including the requirements IDs.

Req#1 Time server periodically sends its time to all the computer after every 10 seconds
Req#2 Receiving the time each computer sends back its clock value to the time server
Req#3 Time server has a priori knowledge to approximate the propagation time (assumed 1 millisecond) and readjusts the clock value
Req#4 Takes a fault-tolerant average of the clock values of all computer including its own
Req#5 Calculated average time is the current time and all the clock should be readjusted the time
Req#6 Time server can generate the reports for each computer by plotting the time vs deviation graph
Req#7 The graphs of computers are deviated more than 15 milliseconds are sent to admin for repair.

Those requirements are designed by UML, sequence, activity etc diagram and developed by C programming language. In that software, time server uses Barkley active time server algorithm for time synchronization. There is a fault tolerance average value for calculating the real time. All the seven requirements are parsed and calculated the term document matrix for each requirement. Using the RDCC methodology presented in Section III, SRS documents, design diagrams and source codes are also parsed and assigned priority values to every module.

TABLE II presents the subtotal priority value of SRS documents, Design diagrams; source codes and finally calculates the weighted sum of requirements priority. According to Equation (4), the weights are assigned to the SDLC phases values. In this case, same weights have been assigned to every phase of SDLC values, because all phases are equal important. The weighted sum is calculated by adding three subtotals collected from requirements, design and source code. According to TABLE II, RDCC# 1 has the highest priority 0.4785 and RDCC# 7 has the lowest priority 0.3696. The values of the weighted sum of requirements RDCC# 1, 2, 3, 4, 5, 6, and 7 are 0.4785, 0.4158, 0.396, 0.4455, 0.3993, 0.4653, 0.3696 respectively. So, the sequence of the prioritized requirements is RDCC# 1, 6, 4, 2, 5, 3, 7.

TABLE III presents the RDCC Test Cases (TC) mapping resolution matrix where the RDCC# ID and their assigned test cases have been pointed. The cell values are the assigning relationship among RDCC# ID and TC. The "Y" value represents the relationship among RDCC ID and TC. According to that table, TC# 1 and TC# 4 are related to RDCC# 1 and TC# 2 is related to RDCC# 2. Based on previous knowledge from TABLE II, it can be concluded that the TC# 1 and TC# 4 should be executed earlier than TC# 2. TABLE IV and V present the prioritized RDCC IDs and prioritized Test Cases IDs respectively. This sequence of test cases can detect the faulty software modules earlier.

978-1-4799-6288-4/14/$31.00 ©2014 IEEE

TABLE II. RDCC MODULE PRIORITIZATION WITH WEIGHTED SUM OF ACADEMIC TIME SYNCHRONIZATION PROJECT (ATSP)

	SRS Documents			Design Documents			Source Code			
RDCC# ID	Values(R)	Weight ()	Total	Values(D)	Weight	Total	Values(C)	Weight	Total	Weighted Sum
RDCC# 1	0.25	0.33	0.0825	0.2	0.33	0.066	0.225	0.33	0.07425	0.4785
RDCC# 2	0.16	0.33	0.0528	0.1	0.33	0.033	0.2	0.33	0.066	0.4158
RDCC# 3	0.1	0.33	0.033	0.1	0.33	0.033	0.1	0.33	0.033	0.396
RDCC# 4	0.15	0.33	0.0495	0.2	0.33	0.066	0.15	0.33	0.0495	0.4455
RDCC# 5	0.11	0.33	0.0363	0.1	0.33	0.033	0.1	0.33	0.033	0.3993
RDCC# 6	0.21	0.33	0.0693	0.2	0.33	0.066	0.2	0.33	0.066	0.4653
RDCC# 7	0.02	0.33	0.0066	0.1	0.33	0.033	0.025	0.33	0.0085	0.3696

TABLE III. TEST CASE RDCC MAPPING MATRIX OF ACADEMIC TIME SYNCHRONIZATION PROJECT (ATSP)

RDCC# ID	TC# 1	TC# 2	TC# 3	TC# 4	TC# 5	TC# 6	TC# 7	TC# 8	TC# 9	TC# 10
RDCC# 1	Y			Y						
RDCC# 2		Y								
RDCC# 3						Y			Y	
RDCC# 4			Y							
RDCC# 5							Y			
RDCC# 6								Y		Y
RDCC# 7					Y					

TABLE IV. PRIORITIZED RDCC ID OF ACADEMIC TIME SYNCHRONIZATION PROJECT (ATSP)

Prioritized RDCC ID	1	6	4	2	5	3	7

TABLE V. PRIORITIZED TEST CASES ID OF ACADEMIC TIME SYNCHRONIZATION PROJECT (ATSP)

Prioritized Test Case ID	TC 1	TC 4	TC 8	TC 10	TC 3	TC 2	TC 7	TC 6	TC 9	TC 7

V. CONCLUSION AND FUTURE WORK

This paper introduces a TCP framework named RDCC which offers combined information from SRS, design documents, and source codes. This framework parses words from different phases of SDLC, and weights main activities in terms of their occurrences and dependences. In SRS, words are parsed and the document-term matrix is generated based on those words frequency and inverse frequency to prioritize requirements ID. Design diagrams are extracted as readable XML files and calculated the interconnectivities among the activities. Source codes are analyzed as call graphs and their interdependency of each class is measured. All priorities from those three steps are added including their weights. A case study was performed to justify the effectiveness of the proposed framework. A future challenge evolving from this research may be to design the prioritized constant for platform independent source codes.

ACKNOWLEDGMENT

This research has been supported by The University Grant Commission, Bangladesh under the Dhaka University Teachers Research Grant No-Regi/Admn-3/2012-2013/13190.

REFERENCES

[1] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[2] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test case prioritization: an empirical study," in *IEEE International Conference on Software Maintenance, (ICSM '99)*, 1999, pp. 179–188.

[3] H. Mei, D. Hao, L. Zhang, L. Zhang, J. Zhou, and G. Rothermel, "A static approach to prioritizing junit test cases," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1258–1275, 2012.

[4] R. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed. New York, NY, USA: McGraw-Hill, Inc., 2005.

[5] I. Sommerville, *Software Engineering (7th Edition)*. Pearson Addison Wesley, 2004.

[6] J. A. Jones and M. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," *IEEE Transactions on Software Engineering*, vol. 29, no. 3, pp. 195–209, 2003.

[7] C. Catal and D. Mishra, "Test case prioritization: a systematic mapping study," *Software Quality Journal*, vol. 21, no. 3, pp. 445–478, 2013.

[8] M. Salehie, S. Li, L. Tahvildari, R. Dara, S. Li, and M. Moore, "Prioritizing requirements-based regression test cases: a goal-driven practice," in *15th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 2011, pp. 329–332.

[9] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[10] M. Arafeen, H. Do *et al.*, "Test case prioritization using requirements-based clustering," in *Sixth International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2013, pp. 312–321.

[11] P. R. Srivastva, K. Kumar, and G. Raghurama, "Test case prioritization based on requirements and risk factors," *ACM SIGSOFT Software Engineering Notes*, vol. 33, no. 4, p. 7, 2008.

[12] H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *International Symposium on Empirical Software Engineering*. IEEE, 2005, pp. 10–pp.

[13] A. Srivastava and J. Thiagarajan, "Effectively prioritizing tests in development environment," in *ACM-SIGSOFT Software Engineering Notes*, vol. 27, no. 4, 2002, pp. 97–106.

[14] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, "The effects of time constraints on test case prioritization: A series of controlled experiments," *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 593–617, 2010.

[15] M. K. Ramanathan, M. Koyuturk, A. Grama, and S. Jagannathan, "Phalanx: a graph-theoretic framework for test case prioritization," in *Symposium on Applied Computing*. ACM, 2008, pp. 667–673.

[16] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," in *Soviet physics doklady*, vol. 10, 1966, p. 707.

[17] N. Kaushik, M. Salehie, L. Tahvildari, S. Li, and M. Moore, "Dynamic prioritization in regression testing," in *4th Int. Conf. on Software Testing, Verifi. and Validation Workshops (ICSTW)*. IEEE, 2011, pp. 135–138.

[18] C. Aggarwal and C. Zhai, "A survey of text clustering algorithms," in *Mining Text Data*. Springer US, 2012, pp. 77–128.