Towards Regression Testing Constraints

Bouchaib Falah and Souhail Marghabi

***Abstract*—Regression testing is one of the most crucial and expensive testing process used to validate modified software and detect new faults introduced by into previously modified and tested code. To reduce the cost of regression testing, software testers may choose to prioritize their test cases to run the "important" ones, chosen by some metrics and constraints, earlier in the testing process. Various approaches have been introduced by previous researches in the form of prioritization technique focusing on specific goals of regression testing. One goal of prioritization is to increase a test suite's rate of fault detection. In this context, previous studies have shown that several prioritization techniques can significantly improve rate of fault detection, but these studies have also shown that the effectiveness of these techniques is relative to the assumptions made by the researchers concerning the testing environment. The variations observed in these experiments have mostly been linked to the subject program, test suites characteristics. This makes it difficult for testers to appropriately choose the correct prioritization technique for their testing scenarios. In this paper, regression prioritization techniques are described. Their performance, in fault detection rate, is assessed and the metrics used to assess the effect of variation through discussing experiments done in this context are specified. Then, the results are analyzed and insight about prioritization techniques selection under these constraints is provided.**

***Index Terms*—Test case, prioritization, regression, Software testing, empirical studies.**

I. Introduction

Software testing is the process of determining if a program behaves as expected. It is an intellectually challenging activity aimed at evaluating the capability of a program or system to determine whether or not it meets requirements [1]. It is an activity that should be done throughout the whole development process [2]. It occurs in each phase of development life cycle, from requirements engineering through delivery and maintenance.

Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of outdated components. Such modifications may cause the software to work incorrectly and affect the overall system functioning. Regression testing works within this framework. It is used to evaluate the modifications of the software. Test suites are generated and executed to ensure that no new bugs have been introduced into previously tested software. Many approaches and techniques for improving the cost-effectiveness of this activity have been investigated.

Manuscript received May 30, 2014; revised August 20, 2014.

The authors are with School of Science & Engineering, Al Akhawayn University in Ifrane, Morocco (e-mail: b.falah@aui.ma, s.marghabi@aui.ma).

These approaches and techniques are categorized as:

1) Retest all;
2) Regression Test Selection;
3) Test Case Prioritization;
4) Hybrid Approach.

Enable testers to order test cases according to priority. Yet, test cases with the highest priority are executed earlier in the regression testing process than those with lower priority. As stated before, test case prioritization techniques help engineers execute regression tests in an order that achieves testing objectives earlier in the testing process. Most researches have focused on the rate of fault detection which is defined as the ability of covering maximum number of faults by a specific test order.

In this context, numerous research studies have discussed the potential of specific prioritization techniques in finding faults with a faster rate; their techniques have focused more on prioritization through data flow information, code coverage, fault exposure potential. Other research literatures [3]-[6] have focused on producing comparative studies of numerous prioritization techniques. However, the main problem of these research studies is that they assume a constant pool of test cases with non-changing coverage during the regression testing process, and therefore they work with a fixed prioritized test suite; that is why empirical studies concluded that prioritization techniques' fault rate performance varied in each subject test program. Additionally, test cases and their coverage metrics may change during regression testing due to modifications of software artifacts.

Therefore, a constraint-aware approach is pursued in this paper. Using data obtained from previous applications of several prioritization techniques to several subject programs, we would compare the performance of these prioritization techniques through their fault detection ratio. The results would be associated together with a technique selection approach incorporating the sources of variations affecting the effectiveness of each prioritization techniques; variations include changes in test programs, test suite size, and characteristics. A cost-benefit decision approach is introduced to allow testers to increase their confidence in selecting the appropriate technique under the selected goal constraint (Fault detection efficiency).

II. Prioritization Technique Background

A. Test Case Prioritization Definition

Test case prioritization problem is defined as follow [7]:

Given: T a test suite, PT , the set of permutation of T , and f a function from PT to the real numbers.

Problem: Find $T' \in PT$ such that

$$(\forall T'')(T'' \in PT)(T'' \neq T')\left[f(T') \geq f(T'')\right] \quad (1)$$

PT represents the set of all possible prioritizations (orderings) of T, and f is a function (heuristic), that applies to any such ordering.

There are several factors affecting the effectiveness of the test case prioritization problem that are not addressed in previous literature [3], [4], [7], [8]. One of them is the range of prioritization goals, which is described as follow:

1) Testers' objective is to increase the rate of fault detection of a test suite
2) Testers may wish to increase the code coverage of the system under test at a faster rate.
3) Testers may wish to ensure the reliability of the system under test at a faster rate.
4) Testers may wish to increase high-risk fault detection rate early in the testing process.
5) Testers may wish to increase the ability of revealing fault related to specific code changes in the regression testing process.

This paper focuses on the evaluation of prioritization techniques based on their effective rate of fault detection. To quantify this goal, a set of metrics will be discussed and explored in the experiment analysis section.

III. Prioritization Using Rate of Fault Detection

As stated in the previous section the goal specified for this study of regression testing is to improve the ratio of revealed faults at early stages in the testing process.

A high rate of fault detection during regression testing would optimize the debugging process for software testers. Similarly, it would also provide faster decision making concerning the overall system under test, allowing strategic decisions to be made earlier than might otherwise be possible.

In this paper, eight different [9] test case prioritization techniques are considered and are classified in four main techniques as illustrated in Fig. 1.

Fig. 1. Classification of test case prioritization.

The first two techniques serve as experimental controls. The last six techniques aim to use test coverage information to prioritize test cases for subsequent execution. These techniques are listed as follow:

A. Comparator Techniques

T1: Random ordering: Where the test cases in test suite are randomly prioritized.

T2: Optimal ordering: Where the test cases are prioritized to optimize rate of fault detection. It represents the ideal case when all faults are known as well as the test cases that reveal each.

B. Statement Level Techniques

T3: Total statement coverage prioritization: This technique prioritizes test cases according to the total number of statements they execute. An example is illustrated in Fig. 2.

```
   procedure P
1. s1
2. while (c1) do
3.    if (c2) then
4.       exit
      else
5.       s3
      endif
6.    s4
   endwhile
7. s5
8. s6
9. s7
```

STATEMENT COVERAGE

STATEMENT	Test Case 1	Test Case 2	Test Case 3
1	x	x	x
2	x	x	x
3		x	x
4		x	
5			x
6			x
7	x		x
8	x		x
9	x		x

Fig. 2 . Procedure p and statement coverage of p achieved by three test cases.

T4: Additional statement coverage prioritization: Focuses on coverage attained to focus on statements not yet covered. This technique uses greedy search algorithm to select a test case that has the greatest statement coverage and then iterates until all statements are covered by at least one test case.

C. Function Level Techniques

T5: Total function coverage prioritization: Prioritizes these test cases according to the total number of functions they cover simply by sorting them in order of total function coverage achieved

T6: Additional function coverage prioritization: it is similar to Additional statement coverage prioritization with only difference that instead of statements, it is considering function level coverage.

IV. Related Work

Regression test selection techniques attempt to reduce the cost of software testing by selecting a subset of an existing test suite for execution in a modified program, ignoring test cases that cannot or are unlikely to reveal errors caused by or revealed through the specific software changes. To improve the effectiveness of regression test selection, numerous researchers have proposed a variety of test case selection methods. Rothermel *et al*. [10], [11], Hutchins *et al*. [12], and Kim and Porter [13] examined different selection test techniques such as minimization, safe, dataflow, control flow, ad hoc and random testing, focusing on their abilities to reduce the cost of testing by decreasing the number of test cases while still detecting the maximum number of defects.

Numerous prioritization techniques have been described in the research literature. Elbaum [14] and Rothermel [3] have shown that at least some of these techniques can significantly increase the rate of fault detection of test suites in comparison to the rates achieved by unordered or randomly ordered test suites. However, no analysis has been done to provide an effective selection approach.

Rothermel *et al*. [8] and Elbaum [4], provide the first formal definition of the prioritization problem and present metrics for measuring the rate of fault detection of test suites. They define prioritization techniques including all of those described in our previous section and provide results of several empirical studies.

Rummel and Kapfhammer [15] investigated whether the all-DUs test adequacy criterion can be used to prioritize the execution of a regression testsuite. They present the "*Instrument and Enumerate"* algorithm that introduces test coverage monitoring and cost reduction. However, the algorithm's provided variant efficiency.

Alspaugh [5] investigates the effectiveness of prioritization search algorithm (knapsack) in reordering test suites with a specific time constraint. This paper presents an example of test case prioritization under variant constraints such as test execution time. However, they did not account for other environment metrics, and the algorithm efficiency still depended on other observed variances.

Among the papers described above, only a few [3], [5], [14] report results of studies or experiments explicitly assessing the ability of prioritization techniques to improve rate of fault detection, relative to each other or to random prioritized test suites. None of the techniques investigated have been absolutely superior to others in all situations. Thus, choice of prioritization techniques needs to consider the dynamic properties of test suites as well as the subject program under test characteristics.

V. Empirical Study Analysis

In this section, the analysis framework adopted for this research is described. The goal of the study is to understand the effect of testing environment variations on the performance of prioritization techniques. First, the measures used to quantify the effectiveness of these approaches as well as evaluating the variant constraints are defined. Next, a list of data retrieved from the similar experiments is presented. Afterwards an analysis over the information inferred from the computed metrics with regards to variation effect and sources in the case of fault detection rate across different subject programs is provided.

A. Dependant Variables

Based on literature [3], the Average Percentage of Faults Detected (APFD) is identified as the main metric that can be used to compare the effectiveness of each regression prioritization technique. APFD measures the average cumulative percentage of faults detected over the course of executing the test cases in the test suite in a given order; higher APFD means higher fault detection rate at early.

Let *TFi* be the order of the first test suites, *Fi* being the number of faults in each test suite Ti. The equation is as follow:

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

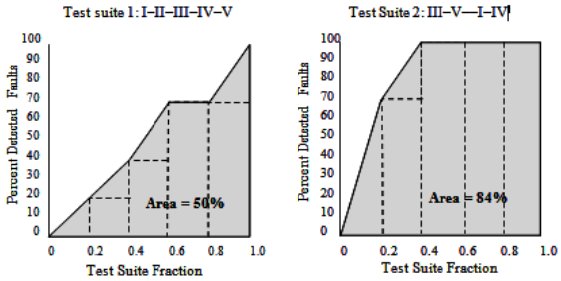
B. Illustration Example

Consider a program with a test suite of five test cases, I through V, such that the program contains ten faults detected by those test cases, as shown in the table of Fig. 3. Consider two orders of these test cases, order T_1: I–II–III–IV–V, and order T_2: III–V–II–I–IV.

Test	Fault									
	1	2	3	4	5	6	7	8	9	10
I	x				x					
II	x				x	x	x			
III	x	x	x	x	x	x	x			
IV					x					
V								x	x	x

Fig. 3. Test suites and faults exposed.

Fig. 4. A and 4. B show the percentages of faults detected versus the fraction of the test suite used, for these two test suites, respectively.



A. APFD for prioritized test suite T1 B. APFD for prioritized test suite T2

Fig. 4. APFD value results for T_1 & T_2.

After running test case I, two of the ten faults are detected; thus 20% of the faults have been detected after 0.2 of the overall test suite T_1has been used. After running test case B, two more faults are detected and thus 40% of the faults have been detected after 0.4 of the test suite has been used. Case B, is different in the sense that the order of the test cases covers all fault at an early fraction of the test suite(APFD = 84%).

C. Independant Variables

The independent variables in our case study are the parameters of the testing process that affect the efficiency and choice of the prioritization technique [14]. These include: the subject programs, the prioritization techniques property, and the changes in the program in a specific version, and the test suites characteristics.

As shown in Table I, a sample list for this research is chosen.

VI. Data Analysis

A. Test Subject Program

To avoid test case generation dependency, we used 7 programs, written in C, with faulty versions and test cases assembled by researchers at Siemens Corporate Research for a study of the fault detection capabilities of control-flow and data-flow coverage criteria [1].

The Siemens programs perform a variety of tasks: tcas is an aircraft collision avoidance system, schedule2 and schedule are priority schedulers, **tot_info** computes

statistics given input data **print_tokens** and **print_ tokens2** are lexical analyzers, and replace performs pattern matching and substitution.

TABLE I: INDEPENDENT VARIABLE METRIC SUBSET

Metric	Description
P_ CH_ L	Percentage of changed line of code
AP _FET	Mean Percentage of functions executed by test
PRG.AN_PATHS	Mean number of paths in the control flow-graph of a function over all functions
P_ TRCHF	Percentage of tests reaching a changed function
A_ TESTS _ CHF	mean number of tests going through changed functions
P_ CH_ INDEX	probability of executing changed functions
P_FCH_ C	percentage of functions changed and covered

The researchers provided large test pools for each of the seven programs; from 100 sample test suites, an average of 7 cases per suite. At the end of the subject program analysis, we ended up with a 41 versions for each base program, variant number of test cases with the specified code coverage criteria [4]. Test Pools were generated using a combination of white-box/black-box techniques focusing on coverage criteria. The characteristics of the overall subject program is described in Table II.

TABLE II: ANALYSED SUBJECT PROGRAM CHARACTERISTICS

Program	Lines of Executable Code	Number of Version	Number of Mutants	Test Pool Size	Average Test Suite Size
Print-tokens	402	7	4030	4130	16
Print-tokens2	483	10	4346	4115	12
replace	516	32	9622	5542	19
schedule	299	9	2153	2650	8
schedule 2	297	10	2822	2710	8
tcas	138	41	2876	1608	6
tot-info	346	23	5898	1052	7

B. Comparative Study Using APFD

To avoid redundant efforts, the extended data of previous literature has allowed us to assemble sample test cases [8] and prioritize test suites with APFD computations which were done manually for each technique and base program. Noting that we can have multiple APFD values for the same applied technique; thus, we represent the results in box plot diagram showing minimum and maximum APFD thresholds for each technique [1].

Fig. 5 illustrates the APFD values of the nine categories of prioritized test suites for each basis program and the overall program total. T_1 is the control group (random) and T_2 is the optimal prioritization group. Comparing the boxplots of T_2 to T_1, we observe that optimal prioritization greatly improved the rate of fault detection (i.e., increased APFD values) of the test suites in comparison to random prioritization. Examining the boxplots of the other prioritization techniques, T_3 through T_6, they all seem to produce considerable rates of fault detection.

At this point, one could easily be misled when making decisions over which technique to adopt first. Still, we notice that we cannot make a general assumption over the "best" technique to use for fault error rate. For instance, the mean value for T_3 in **print_token** was higher 20 points than the case in schedule 2; T_5 had a high APFD value in **print_token2**, better than T_6 and T_4, but it was lower than mean in **tcas**.

The summary of the discussed findings is illustrated in Fig. 5.

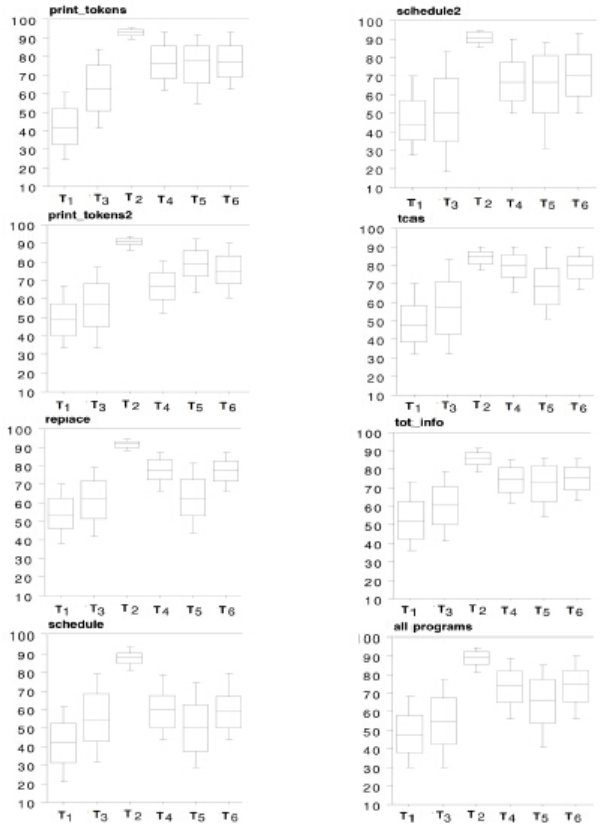


Fig. 5. Box plot representation for a range of APFD values for prioritization techniques, T_1(random), T_2(optimal), T_3(total_stat), T_4(add_stat), T_5(total_functional), T_6(Add_functional).

VII. COST-BENEFIT TRADE-OFF AND DECISION APPROACH

Since the purpose of this analysis is providing test engineers with methodologies and tools to appropriately choose prioritization techniques, we discuss an improved selection approach which attempts to recover from the weaknesses of the APFD comparison method and we provide more efficient prioritization selection framework.

Priority cost model approach introduced in previous literatures is used because the cost factor has been considered in the decision making process [16]. The higher the APFD, the more efficient the testing process is.

Cost-Benefit Approach:

Hence, we could quantity by percentages the APFD

advantage of using one technique over another. The approach includes:

1) Keep track of the APFD (minimum/maximum/median) computed for each subject program and prioritization techniques;
2) Compute the number of occurrences when a specific technique has a higher APFD than another
3) Set an acceptance threshold for which it is implied that the superior yielding technique is to be selected.

Focusing on prioritization technique constraints, we chose to use the cost model approach by evaluating the chosen goal for regression testing; in this case, the ratio of fault detection efficiency of test suites. Hence, we investigate the empirical study APFD results and compare the techniques based on the metric. We define a decision threshold range from 0% to 25%. In general, the approach starts by comparing two techniques, (e.g. T_1 *and* T_2) and calculate the percentage of occurrences where T_1's APFD exceeds T_2 by the defined threshold range in all the APFD computations. For instance, in case 4, we notice that with threshold 5%, T_5 (total functional coverage) performs 75% better than T_1 (Random Ordering).

Applying the suggested approach we can generate outputs as shown in Table III.

TABLE III: REGRESSION PRIORITIZATION TECHNIQUE SELECTION

Case	Techniques Compared	RPTS		
		0%	5%	25%
1	T_1 vs. T_2	11	5	2
2	T_1 vs. T_3	35	2	2
3	T_1 vs. T_4	16	10	0
4	T_5 vs. T_1	25	75	79
5	T_1 vs. T_6	42	11	5
6	T2 vs T3	50	79	30
7	T2 vs T4	68	30	5
8	T2 vs T6	63	46	14
9	T2 vs T5	37	65	20
10…	T5 vs T4	50	55	12

One interesting observation is that some techniques' performance advantage decreases as the threshold, expected gained benefit, increases; example of T_5 vs T_4, where T_5 loses considerable performance benefit if we expect a 25% advantage threshold.

Finally, although we cannot claim that the cost-benefit trade-off approach presented can be generalized to other programs with different versions, programs, and test suites; we expect further experimentation, where we will integrate the independent variable metrics into our priority technique selection approach, to provide testers with more confidence in their regressing technique approach.

VIII. CONCLUSION

In this paper, a study of eight test case prioritization techniques applied across eight systems was presented. Although previous studies of test case prioritization [3], [8] have been conducted in similar settings, the set of subjects we have considered (7 programs) resulted in a considerable testing pool. These techniques were compared based on their ability to improve the rate of fault detection of test suites; this as being only one of the numerous goals, testers might chose as listed in section I.

The results of this analysis have shown that prioritization techniques performance is subject to variations related to the test suite characteristics, changes of subject programs, and functions. It has also confirmed the previous findings derived from different empirical studies [3], [4], [14]. Thus, the need for a prioritization technique selection approach is of considerable importance to the testers' community.

The basic RPTS (Regression prioritization technique selection) approach introduced in section VI, represents a practical framework providing testers with the ability to select the optimal technique, with higher confidence, while accounting for cost-benefit trade-offs.

IX. FUTURE WORK

We would like to extend our cost-benefit approach to account for the characteristics of test case scenarios specifications, such as the different variances described in previous sections. Hence, we will consider quantifying the independent experimental variables defined in Table I, and through a decision-tree representation integrate these metric values with our decision-making threshold mechanism to increase the ability of the approach the appropriate prioritization technique with higher efficiency. Additionally, we are considering extending the scalability of the discussed approach to larger industrial application while accounting for the bias factor that might affect the validity of our results.

REFERENCES

[1] B. Falah, K. Magel, and O. E. Ariss, "A complexity based regression test selection strategy," *Computer Science & Engineering: An International Journal (CSEIJ)*, vol. 2, no. 5, October 2012.

[2] A. Bertolino, "Software testing," *IEEE SWEBOK Trial Version 1.00*, May 2001, ch. 5.

[3] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization," in *Proc. the International Conference on Software Engineering*, pp. 329–338, 2001.

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

[5] S. Alspaugh and G. M. Kapfhammer, "Efficient time-aware prioritization with knapsack solvers," in *Proc. the ASE 2007 Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, pp. 13-18, Atlanta, Georgia, November 2007.

[6] G. M. Kapfhammer and M. L. Soffa, "Using coverage effectiveness to evaluate test suite prioritizations," *Weaseltech*, 2007.

[7] H. Leung and L. White, "Insights into Regression Testing," in *Proc. the Conference on Software Maintenance,* pp 60-69, Oct. 1989.

[8] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test case prioritization: an empirical study," in *Proc. the International Conference on Software Maintenance*, pp. 179–188, 1999.

[9] R. Gregg, H. U. Roland, C. Y. Chu, and H. M. Jean, "Prioritizing test cases for regression testing," *CSE Journal Articles*. pp. 9, 2001.

[10] T. L. Graves, M. J. Harrold, J. Kim, A. Poiler, and G. Rotliennel, "An empirical study of regression test selection techniques," *ACM Transactions on Software Engineering and Methodology*, vol. 10, no. 2, pp. 184-208, 2001.

[11] G. Rothermel and M. J. Hanold, "A safe, efficient regression test selection tecirniqite," *ACM Transactions on Software Engineering and Methodology*, vol. 6, pp. 173-210, 1997.

[12] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and controlfiow-based test adequacy

criteria," in *Proc. the 22ndInternational Conference on Software Engineering*, 1994, pp. 191-200.
[13] J. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environnents," in *Proc. the tile 24th International Conference*, pp. 19-25, 2002.
[14] S. Elbaum, D. Gable, and G. Rothermel, "Understanding and measuring the sources of variation in the priori-tization of regression test suites," in *Proc. the Seventh International Software Metrics Symposium*, Institute of Electrical and Electronics Engineers, Inc., vol. 2, no. 9, 2001..
[15] M. J. Rummel and G. M. Kapfhammer, "Towards the prioritization of regression test suites with data flow information," in *Proc. the ACM Symposium on Applied Computing*, 2005, pp. 1499-1504.
[16] A. Malishevsky, G. Rothermel, and S. Elbaum, "Modeling the cost-benefits tradeoffs for regression testing techniques," in *Proc. the International Conference on Software Maintenance*, pp. Oct. 2002.

Bouchaib Falah was born in Casablanca, Morocco in 1966. He received his Ph.D in software engineering from North Dakota State University, U.S.A, in 2011 and a master degree in computer science from Shippensburg University, Pennsylvania, U.S.A, in 2001, and a bachelor degree/teaching certificate from Ecole Normale Superieure, Casablanca, Morocco in 1990. He has more than 20 years of combined experience in developing and implementing computer science and technical math curriculum for different colleges and universities as well as web designer for multimillion-dollar organizations in USA and researcher in different projects, he is currently an assistant professor at Al Akhawayn University, teaching graduate and undergraduate software engineering courses, School of Science and Engineering. Besides teaching high school level math in Morocco and college mathematics and computer science at Harrisburg Area Community College in Pennsylvania, Suny Orange Community College in New York, Pennsylvania State University in Pennsylvania, Central Pennsylvania College in Pennsylvania, Concordia College in Minnesota, and North Dakota State University in North Dakota, he has an extensive industrial experience with Agri-ImaGIS, Synertich, and Commonwealth of Pennsylvania Department of Environmental Protection. His current research interests include Complexity Metrics, Security testing, agile methodology and extreme programming, mutation testing, regression testing, software engineering processes, web application testing, and design and architecture. He published a book titled: An Approach to Regression Test Selection Based on Complexity Metrics, Scholar's Press as well as many journal papers on software testing.

In 2014, Dr. Falah became a member of APCBEES and in 2013, he was awarded certificates from International Association of CS and IT as well as ARPN journal of systems and software.