IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 20, 2017, accepted August 25, 2017, date of publication September 7, 2017, date of current version September 27, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2749502

Regression Testing of Database Applications Under an Incremental Software Development Setting

RAÚL H. ROSERO[1,2], OMAR S. GÓMEZ[2], AND GLEN RODRÍGUEZ[1], (Member, IEEE)
[1]Universidad Nacional Mayor de San Marcos, Lima 4559, Peru
[2]Escuela Superior Politécnica de Chimborazo, Riobamba 060155, Ecuador

Corresponding author: Raúl H. Rosero (raul.rosero@unmsm.edu.pe)

ABSTRACT Software regression testing verifies previous features on a software product when it is modified or new features are added to it. Because of the nature of regression testing it is a costly process. Different approaches have been proposed to reduce the costs of this activity, among which are: minimization, prioritization, and selection of test cases. Recently, soft computing techniques, such as data mining, machine learning, and others have been used to make regression testing more efficient and effective. Currently, in different contexts, to a greater or lesser extent, software products have access to databases (DBs). Given this situation, it is necessary to consider regression testing also for software products such as information systems that are usually integrated with or connected to DBs. In this paper, we present a selection regression testing approach that utilizes a combination of unsupervised clustering with random values, unit tests, and the DB schema to determine the test cases related to modifications or new features added to software products connected to DBs. Our proposed approach is empirically evaluated with two database software applications in a production context. Effectiveness metrics, such as test suite reduction, fault detection capability, recall, precision, and the F-measure are examined. Our results suggest that the proposed approach is enough effective with the resulting clusters of test cases.

INDEX TERMS Software regression testing, fault detection capability, test suite reduction, software verification, software engineering.

I. INTRODUCTION

Software regression testing (RT) plays an important role in the software development cycle [1], as it helps to verify that faults are not injected into previous versions of a software product when new features or existing modifications are performed. Executing the total set of test cases is an expensive and laborious endeavor, therefore determining a subset of test cases to be executed in a software regression test is an active topic of research. To this end, different approaches have been studied and applied, including minimization, prioritization, selection, and, recently, optimization of test cases [2]. Where in this last approach, are included strategies for the determination of the group of test cases used in a regression testing [3] such as: greedy algorithms for minimization [4], graph-based relations for selection [5], capacity-based fault detection for prioritization [6], and fuzzy entropy-based optimization [7], among others.

The cost of running a regression test is related to the size of the product to be verified and the suite of test cases to be executed, this can cause limitations in the amount of computational resources available for this purpose [8], [9].

If we consider iterative and incremental development environments in which the software is developed and delivered in short cycles, the execution of the complete suite of test cases is an unacceptable practice given the volume of test cases to execute (the goal of pursuing agility is lost). This issue has raised the study of soft computing approaches [10], in conjunction to agile approaches [11] in order to identify test cases that guarantee an acceptable level of verification of the software product [12].

Results of recent studies indicate that a more effective alternative for optimizing regression tests is to cluster the test cases according to some criterion, pattern or characteristic [10], [13]–[15], such that test cases that detect the same fault are in the same cluster. Examples of these strategies include test execution profiles [16], [17], histories of test runs [14], function calls [18], partitioning programs, filtering partitions [19], access to databases (DBs) [20]

and sampling the most representative test cases for each cluster [21]. However, the problem to solve in these approaches is to determine the quantity and size of the clusters in order to balance the cost and the effectiveness of the approach.

In the field of machine learning, cluster analysis is a technique for training and organizing data according to mathematical approaches independent of the application domain [22]. Sometimes it is necessary to have additional information in addition to the data, such as the number and sizes of the clusters to be formed, which makes the process expensive and sometimes limits the procedure; consequently, supervised clustering requires experience to manually group such information [23]. An alternative is to use unsupervised clustering, in which probabilistic approaches are used to determine the number of clusters [24].

Conventionally, regression testing (RT) techniques have been applied from the perspective of the code, i.e., considering the software product as a finite state machine [2], keeping the data constant, which does not take into account their concomitant evolution [20]. However, in software products with access to DBs, two types of states are simultaneously observed (code states and data states); thus, RT techniques should consider these two aspects. At one side, the state of the code is organized as a set of labeled memory locations at which the individual data is stored (code states). However, at the other side, the state of a DB is organized according to different data models, such as the relational model. Moreover, large volumes of data can be affected by a single code instruction that can affect the behavior of other instructions in separated components of the software product that access a DB [25].

In this work, we present an approach to RT for software products with DB access under an incremental software development context. It is based on the clustering of the DB access code along with the DB schema. Based on an analysis of code that accesses the DB, our proposal uses an unsupervised clustering approach with random values that selects a set of test cases. We conduct an empirical evaluation of our approach with two database software products running in a production setting.

The rest of this document is organized as follows. Section II presents the related work. Section III presents the proposed approach and the corresponding algorithm. Section IV presents the empirical evaluation of the proposal. Section V presents the results. Section VI presents the discussions and threats to validity. Finally, the conclusions are discussed in section VII.

II. RELATED WORK

More than two decades have passed since Rothermel and Harrold [3] introduced the concept of software regression testing. Since then, different approaches to regression testing have emerged. Some of these RT approaches are based on strategies that analyze the source code, strategies that analyze characteristics of the test cases. Recently have been reported combinations of the previous strategies with soft computing techniques [2], [27].

Three well-known approaches for RT are minimization, selection and prioritization. These approaches can be defined as following:

Minimization. Let P be a software product and T the set of test cases associated with P. In minimization, the problem is to obtain a subset of test cases T' that does not include redundant or obsolete test cases to verify P.

Selection. In selection, the strategy is to determine a subset of test cases T' in such a manner that the modifications to the software product P' are tested with T'.

Prioritization. In prioritization, the goal is to determine an ideal permutation of sequences of test cases to improve the performance of the RT technique.

Once described these three approaches, we briefly discuss some relevant and recent approaches to RT reported in the literature, and which are further detailed in [2].

Rothermel and Harrold [26] presented a strategy to analyze and detect modification in control flow graphs of the original and modified programs to determine the test cases related to them. Chen *et al.* [28] presented a technique with regards to the level of the code entities of functions and variables. Leung and White [29] introduced a technique with emphasis at the module level; this strategy determines the modified modules along with their test cases, and it considers them for the integration tests. The abstraction of the program into models has also been applied in [12].

Under the approach of analyzing test cases, it has been observed that some test cases have similar behaviors and that they sometimes are related to the same fault. This situation has motivated more research concerning the classification of test cases into groups, this based on predetermined metrics [30]. We also have found works where software metadata or feature specifications are considered. For example, Vangala *et al.* [16] used execution profiles and static executions to compare test cases and to apply clustering algorithms. Orso *et al.* [31] used metadata in eXtensible Markup Language (XML) format to select the test cases. Kim *et al.* [32] reported a technique that combines the history of executions with code modifications. Wikstrand *et al.* [33] presented a technique that relates the results of the test cases with the defects found and also with the previous modifications. Ficco *et al.* [34] presented a technique based on the capacity for fault detection of the test cases under the methodology of incremental iterative development.

Because the connections between groups of test cases are sometimes hidden [16], it is necessary to determine patterns that relate these groups. For this purpose, recently optimization approaches for RT have emerged. Techniques such as soft computing [15], data mining [35], beehive networks [36], and machine learning [27], [2] are starting to be applied in the context of RT. As regards data mining and machine learning techniques, a common strategy followed is the identification of clusters. In the context of RT, clustering analysis considers grouping a set of test cases. For example, Dickinson *et al.* [37]

TABLE 1. Summary of software regression testing of database applications using clustering analysis.

Authors	Regression Technique	Analyzed Feature	Clustering Technique Used	Clustering Approach	Application Context
Haraty et al. [42]	Minimization	SQL statement structures	Graph walk , firewall	Not applicable	Academic
Willmor and Embury [25]	Selection	Code association with the DB	Control flow graphs	Not applicable	Academic
Rogstad et al. [47]	Selection	Control flow graph of DB modules at a procedural level	k-means	Supervised	Academic
Rogstad et al. [46]	Selection	UML structures of the test cases	DART Model	Unsupervised	Industrial
Rogstad and Briand [20]	Selection	Test cases deviation patterns in DB manipulations, prioritization of requirements	Expectation maximization	Unsupervised	Industrial

applied data mining to filter and cluster test cases based on their execution and program profiles. Other works [38]–[41] have also reported the application of clustering analysis for conducting software RT.

In the context of software products with data access, Haraty *et al.* [42] considered regression testing from the perspective of stored procedures under a firewall approach. Willmor and Embury [25] presented their technique of safe regression testing using the white-box approach along with graphs representing the code. Tuya *et al.* [45] proposed a series of experiments to evaluate and compare the effectiveness of different white-box techniques with the coverage of SQL (Structured Query Language) statements, with the aim of selecting test cases associated with a database. Rogstad *et al.* [46] considered the transactional aspect of a DB, but from the perspective of black-box models using classification trees based on the similarity of the test cases. Similarly, Rogstad and Briand [20] considered test cases related to a database for grouping deviations from the database (differences between database manipulations and the versions of the program); this approach is based on an entropy criterion to evaluate different clustering techniques [28]. Table 1 shows a summary of the characteristics used in these works.

As shown in Table 1, we found works that propose black-box [25], [42] and white-box approaches [20], [46], [47] of regression testing used in database applications. Concerning approaches that use white-box, we have not found works that consider, under an integral approach, modifications to the source code in conjunction with the accesses to the database and its related test cases.

In summary, we have found some works that discuss regression testing approaches applied to database applications, however, we have not found regression testing approaches considering unsupervised clustering strategies (where a probabilistic criteria is used to determine the number of groups) of database applications developed under an iterative and incremental settings. In addition, we have not found regression testing approaches that consider the database schema in an integral way, along with the database access code and the associated test cases (derived from an unsupervised clustering strategy).

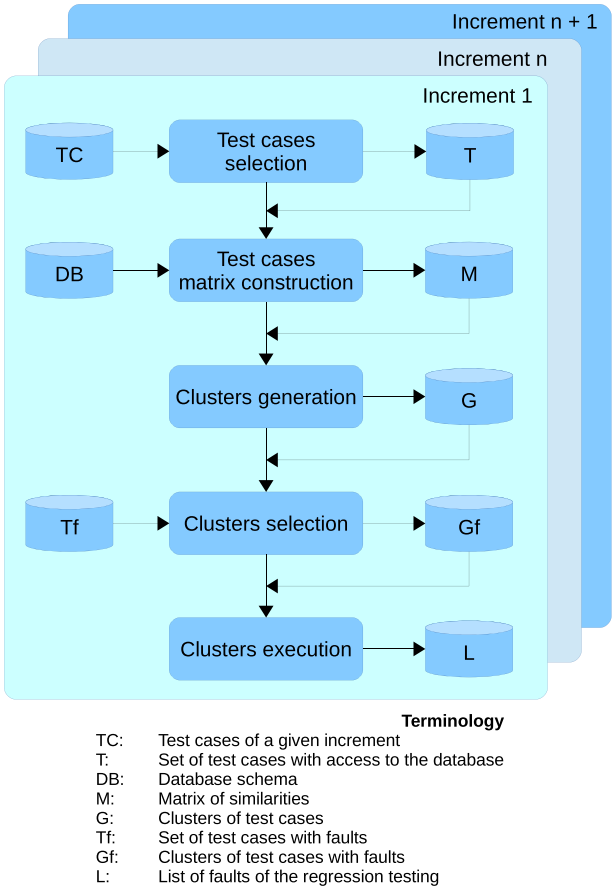

FIGURE 1. Approach for regression testing of database applications.

III. PROPOSED RT APPROACH

In this section, we detail the proposed regression testing of database applications approach and describe the algorithms used.

At a glance, our proposal consists in grouping test cases associated with data manipulation, particularly on the fields in the tables of a DB, and then, in successive development increments, execute the group or groups of test cases that contain the test cases that have detected faults. The proposed approach is shown in Fig. 1.

TABLE 2. Matrix of test cases accessing the DB.

	Tables						
	$Table_1$			…	$Table_n$		
	Col_1	Col_2	Col_i	…	Col_1	Col_2	Col_i
TC_1	1	0	1	…	0	1	0
	.	.	.	....	....	...	...
TC_n	.	.	.		....	...	...

Following, we describe the involved steps of the proposed approach.

1) *Test cases selection* The objective of this initial step is to analyze the code of the test cases associated with a new feature or a modification that contains access to the fields of the tables of a given DB. To do this, a textual analysis of the software product code is performed. The results are the test cases that should be added to the set of test cases (T) that access the DB.
2) *Test cases similarity matrix construction* In this step, we obtain information regarding the table(s) and columns(s) of each test case (T). We consider functionalities such as store procedures, triggers, functions and views. With this information, a binary matrix (M) is constructed. In this matrix, the value 1 represents a DB access operation specified in the test case. This access is related to the attribute(s) of a given entity/relationship, as shown in Table 2. This process yields an M_{mxn} matrix in which the rows (m) correspond to test cases and the (n) columns represent the fields of the DB tables of the software product to be examined.
3) *Test cases clusters generation* In this step, we use an unsupervised clustering algorithm under the strategy of expectation-maximization (EM) [48], which allows the analysis of statistical descriptors of the test cases (mean and standard deviation). Based on this information, we determine a probabilistic distribution function to assign the membership of a test case to a cluster based on the similarity matrix. The algorithm can decide how many clusters to create based on cross-validation of the data; this algorithm uses a finite Gaussian mixture model [49] that assumes that all attributes are independent random variables. After applying this algorithm, we get one or more clusters (G).
4) *Clusters selection* In this step, we select the clusters that are going to be executed for the regression test. We determine in which clusters the new or modified test cases (Gf) are located; we also consider the test cases that revealed faults (Tf).
5) *Clusters execution* In this step, the test cases (Gf) are executed, thus generating a list of faults related to the test cases, if applicable.

A. ALGORITHM USED IN OUR PROPOSAL

Following, we describe the algorithms used in each step of the proposed approach (test cases selection, test cases matrix construction, test cases clusters generation, clusters selection, and clusters execution). Table 3 lists and describes the identifiers used in each of the algorithms designed.

TABLE 3. Identifiers used in the algorithms.

Variable	Description
DB	Database
T	Set of test cases
F	Set of features
Fi	i-th feature, $f_i \in F$
TC_i	Set of test cases from i-th feature $f_i \in F$
tc_i	i-th Test case of a feature
$tc_{i,j}$	i-th Test case $tc_i \in TC_i$ from j-th feature
TestDB	Set of test cases with access to DB
TestDBfailed	Set of test cases failed with access to DB
MatTestDB	Test cases matrix with access DB
MatClusterTestDB	Test cases cluster matrix with access DB
MatClusterRegressionTestDB	Failed matrix of clustered test cases with access to DB
FaultsList	Set of faults for the regression test
NumTestDB	Number of test cases with access to the DB

Our approach consists in performing small regression tests incrementally with each version of the software product, i.e., whenever features (fi) are added or modified in the software product. Algorithm No. 1 presents the basic steps for this.

Algorithm 1 clusterSelectionRegression Test

Input: f_i, TC_i
Output: Faults List

```
1: clusterSelectionRegressionTest process
2: repeat
3:  for each f_i do //fi ∈ F
4:    TestDB ← selectTestDB(f_i, TC_i)
5:    MatTestBD ← buildMatrixTestDB(testDB)
6:    MatClusterTestDB ← clusterizationEM(MatTestBD)
7:     MatClusterRegressionTestDB ← (MatCluster
       TestDB ∩ TestDBfailed)
8:     executeClusterRegressionTestDB
       (MatClusterRegressionTestDB)
9:  end for
10:   until ∄ f_i
11: end clusterSelectionRegressionTest process
```

1) *Main Algorithm* For each feature (fi), a selection of test cases that access the DB is performed, and then, the process of construction of a similarity matrix based on the accesses to the fields of the DB is applied. Once the matrix is built, clustering analysis is applied. Finally, the clusters containing test cases that have failed are executed, and their faults are examined, as shown in Algorithm No. 1.

2) *Test cases selection* For the implementation of this step, we developed a software tool that analyzes the DB schema. This tool is able to determine the DB functions that are invoked from the test case code of the software product. The output of this phase constitutes the test cases that access the DB (TestDB), as shown in Algorithm No. 2.

Algorithm 2 selectedTestDB

Input: f_i, TC_i
Output: TestDB, NumTestDB

```
1: selectedTestDB process
2: numTestDB ← 0
3: repeat
4:     if TC_i access DB then
5:         TestDB ← TestDB ∪ TC_i
6:         NumTestDB++
7:     end if
8: until ∄ TC_i
9: end selectedTestDB process
```

3) *Test cases similarity matrix construction* In this phase, as Algorithm No. 3 illustrates, the similarity matrix is constructed iteratively, with *n* rows, one for each test case that accesses the DB, and *m* columns, which correspond to the attributes of each entity/relationship. In each row, the value of 1 indicates that the test case is related to some attribute; the value is 0 otherwise. The developed tool is also used for this step, as shown in Algorithm No. 3.

Algorithm 3 buildMatrixTestDB

Input: TestDB, numTestDB
Output: MatTestDB

```
1: buildMatrixTestDB process
2: i←1
3: while i < numTestBD do
4:     MatTestDB ← (MatTestDB ∪ (TestDB_i))
5: end while
6: end buildMatrixTestDB process
```

4) *Test cases clusters generation and selection* For the clustering analysis, the use of the Weka data mining tool was considered for practical reasons [50] (owing to its available documentation, an active discussion forum and the existence of application programming interfaces). This tool implements different clustering algorithms. Among these, we selected the probabilistic algorithm EM [48], which has the advantage of determining a *k* number of clusters based on the information of the test cases of the similarity matrix, see Algorithm No. 4.

5) *Clusters execution* Algorithm No. 5 presents the final phase of the approach, which executes the test cases that belong to a certain cluster in which the test cases related to a given feature (fi) has failed (TestDBfailed); the result of this step is a list of detected faults.

Algorithm 4 clusterizationEM

Input: MatTestDB
Output: MatClusterTestDB

```
1: clusterizationEM process
2:   Θ Vector desconocido de parámetros
3:   Θ^0, Θ^1, ...., Θ^T, T criterio de convergencia
4:   T←0
5:   Θ^0 ←0
6:   Repetir
7:     Q(Θ, Θ^t) ← E[log p(x^g, x^m I Θ) I x^g, Θ^t]
8:     Θ^{t+1} ← arg max_Θ Q(Θ, Θ^t)
9:     t ← t+1
10:  until convergence criterion
11: end clusterizationEM process
```

Algorithm 5 executeClusterRegressionTestDB

Input: MatClusterRegressionTestDB
Output: FaultsList

```
1: executeClusterRegressionTestDB process
2:     FaultsList ← ∅
3:   for each Tc_i in TestDBfailed
4:       execute(MatClusterTestDB ∩ TestDBfailed)
5:       FaultsList ← FaultsList ∪ tc_i
6:     end for
7: end executeClusterRegressionTestDB process
```

IV. EMPIRICAL EVALUATION

In this section, we evaluate the proposed approach through an extensive empirical evaluation; for this purpose, we used two database software products running in a production environment.

A. INDICATORS AND METRICS

To evaluate the effectiveness of the proposed approach, the metrics proposed in [51] were considered, such as the ability to reduce the suite of test cases and the ability to detect faults in regression tests. The metrics used in our evaluation are as follows:

Percentage reduction of the suite (TR). Let T be the total number of test cases with access to the DB and T' the number of test cases selected for a regression, then TR is given by equation in (1):

$$TR = \frac{T - T'}{T} \times 100 \quad (1)$$

Precision (P). Let T'f be the set of all test cases selected with access to the DB that reveal faults, then, P is given by

equation in (2):

$$\text{Precision} = \frac{T'f}{T'} \tag{2}$$

Recall (R). Let T'f be the set of test cases that reveal faults. Then, the metric of R is given by equation in (3):

$$\text{Recall} = \frac{|T'f|}{|Tf|} \tag{3}$$

Fault detection capability. This is a metric directly related to recall. Specifically, equation (3) is used to calculate the fault detection capability of the selected test cases.

F-measure (F). It is a combination of two metrics widely used in information science, precision (P) and recall (R); the F-measure evaluates the benefit integrally and is given by equation in (4):

$$\text{F-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

B. EXPERIMENTAL SUBJECTS

For the evaluation of our proposal, we used two database software applications developed in Java, both of which are employed in an academic context. The first one, called "Estafeta", is a software product used to manage the planned weekly working days of professors of an Ecuadorian university.

The second product identified as "Silabo", is used to manage the contents of the courses taught by a teacher at an Ecuadorian university.

Table 4 presents the technical characteristics of each software product.

TABLE 4. Technical characteristics of the software products.

Software Products	Characteristics	
Estafeta	LOC	57,797
	DB	Relational
	Entities	39
	Relationships	92
	Average attributes of entities/ relationships	10
	No. Test cases with access to DB	197
	Development platform	NetBeans
	Libraries	JUnit y DBUnit
	No. of versions (increments)	5
Silabo	No. LOC	35,880
	DB.	Relational
	Entities	42
	Relationships	88
	Average attributes of entities/ relationships	10
	No. Test cases with access to DB	309
	Development platform	NetBeans
	Libraries	JUnit y DBUnit
	No. of versions (increments)	9

C. EXPERIMENTAL SUBJECTS

As previously mentioned, the clustering technique is used to group test cases based on their similarity on the fields of the tables of a DB. For the first product (Estafeta), five versions were incrementally developed, each with their own set of test cases. For the second product (Silabo), nine versions were developed. In both cases, the methodology of incremental iterative development was followed; that is, in each cycle, new features were added, so new test cases were also designed and implemented following the methodology and tools for unit tests under Java for both, the code and the DB (JUnit, DBUnit) [52].

Table 5 presents the new test cases by version for each of the software products, and the cumulative test cases for a given regression test.

TABLE 5. Number of test cases for rach version of the software products.

Software Product	Version	Test cases (Tn)	Test cases for the regression (T)
Estafeta	1	29	-
	2	43	72
	3	62	134
	4	29	163
	5	36	199
Silabo	1	27	-
	2	33	60
	3	29	89
	4	30	119
	5	39	158
	6	49	207
	7	32	239
	8	22	261
	9	39	300

For each of the products, the respective test cases of each of the versions were executed, after which a regression test of the product was performed for each version. Table 6 reports the faults (Tf) that were identified in the regression tests.

TABLE 6. Faults and test cases for each version.

Software Product	Version	Test Cases (T)	Faults (Tf)
Estafeta	2nd	72	8
	3rd	134	11
	4th	163	9
	5th	199	10
Silabo	2nd	60	8
	3rd	89	10
	4th	119	15
	5th	158	5
	6th	207	7
	7th	239	10
	8th	261	5
	9th	300	4

To evaluate the approach, it was assumed that the cost of the test cases is equal, considering the execution times of the test cases, which, on average, required 30 seconds to execute, with slight deviations, in addition to their severity and the fault patterns of the programmers.

For the clustering phase, we used the probabilistic algorithm EM [48], which is an extended version of the K-means in which the estimation of the model parameters is optimized. EM uses the soft assignment method for assigning elements

to a cluster (the element can be clustered into multiple clusters), whereas other algorithms, such as K-means uses hard assignment (an element belongs to a single cluster) [53]. EM determines the parameter k, which specifies the number of clusters through a cross validation of the information supplied (see algorithm No. 4).

Regarding the number of iterations in this work, we use 1×10^{-6} as the minimum value, the maximum number of iterations was 10, and the standard deviation was used to judge for the convergence, thus, following the same approach performed in [49].

It should be noted that in order to reduce the random impact of the random determination of the centroid (this constitutes a weakness in the clustering algorithms), we employed the optimization scheme described in [54] and [55]. We performed a statistical analysis for each version with 10 different seeds (1, 10, 50, 100, 150, 200, 250, 500, 750, 1000), obtaining measurements for each of the variables considered in this study, and later performed a statistical analysis (mean, variance, standard deviation) for each version of each software product.

Tables 7 and 8 present the data obtained for the fourth and seventh versions of the software products (Estafeta and Silabo), respectively. Note that the parameter k was determined using the EM algorithm of the Weka software tool.

TABLE 7. Clustering of the 4th version of Estafeta with different seeds.

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	F-measure
1	3	163	29	148	9	9	9.2%	100%	6%	0.11
10	3	163	29	147	9	9	9.8%	100%	6%	0.12
50	3	163	29	122	9	9	25.2%	100%	7%	0.14
100	4	163	29	33	9	9	79.8%	100%	27%	0.43
150	4	163	29	154	9	9	5.5%	100%	6%	0.11
200	7	163	29	121	9	9	25.8%	100%	7%	0.14
250	6	163	29	134	9	9	17.8%	100%	7%	0.13
500	5	163	29	154	9	9	5.5%	100%	6%	0.11
750	6	163	29	147	9	9	9.8%	100%	6%	0.12
1000	7	163	29	148	9	9	9.2%	100%	6%	0.11

As can be observed from Table 7, for the fourth version of the Estafeta product, a reduction rate of 79.8% was reached. This with a maximum precision of 27% corresponding to four clusters (with the seed set to 100). The lowest values were obtained when the number of clusters was equal to five (seed equals to 500), which yielded a reduction rate of 5.5% and an accuracy of 6%. The average test reduction rate for this version is shown in Table 9, as we see it is equals to 20% (0.20).

In the case of the seventh version of the Silabo product (Table 8), a reduction rate of 87.0% was reached, this with a maximum precision of 29% corresponding to nine clusters (seed set to 1 and 100). The lowest values were obtained when the number of clusters was equal to 6 (seed equals to 1000). The average test reduction rate for this version of the Silabo is equals to 54% (0.54), as shown in Table 10.

TABLE 8. Clustering of the 7th version of Silabo with different seeds.

seed	K	T	Tn	T'	Tf	T'f	TR	R	P	F-measure
1	9	239	32	31	9	9	87.0%	100%	29%	0.45
10	7	239	32	180	9	9	24.7%	100%	5%	0.10
50	3	239	32	195	9	9	18.4%	100%	5%	0.09
100	9	239	32	31	9	9	87.0%	100%	29%	0.45
150	4	239	32	40	9	9	83.3%	100%	23%	0.37
200	7	239	32	170	9	9	28.9%	100%	5%	0.10
250	5	239	32	41	9	9	82.8%	100%	22%	0.36
500	5	239	32	40	9	9	83.3%	100%	23%	0.37
750	7	239	32	171	9	9	28.5%	100%	5%	0.10
1000	6	239	32	203	9	9	15.1%	100%	4%	0.08

TABLE 9. Statistics for each version of the software product Estafeta.

Metric	Mean	SD	Variance	Rank
Version 2				
TC Reduction rate (TR)	0.15	0.12	0.01	0.32
Recall	1	0	0	0
Precision	0.13	0.20	0.00	0.05
F-measure	0.23	0.03	0.00	0.08
Version 3				
TC Reduction rate	0.06	0.05	0.00	0.12
Recall	1	0	0	0
Precision	0.09	0.00	0.00	0.01
F-measure	0.16	0.01	0.00	0.02
Version 4				
TC Reduction rate	0.20	0.22	0.05	0.74
Recall	1	0	0	0
Precision	0.09	0.07	0.00	0.21
F-measure	0.15	0.10	0.01	0.32
Version 5				
TC Reduction rate	0.16	0.27	0.07	0.81
Recall	1	0	0	0
Precision	0.08	0.07	0.01	0.24
F-measure	0.14	0.11	0.01	0.35
Product Average				
TC Reduction rate	0.14	0.19	0.03	0.82
Recall	1	0	0	0
Precision	0.10	0.05	0.00	0.23
F-measure	0.17	0.08	0.01	0.35

V. RESULTS

In this section, we present the results obtained after evaluating the proposed approach, it in terms of the metrics previously defined. Tables 9 and 10 present the average data for each regression test for each version of the software products.

Note that for some metrics, high values were obtained, however these values are balanced with others low values (for example, compare the fourth and third versions [Table 9] in terms of the metric TR, these values are sensitive to k and the seed). The metric of detection capacity (R) was 100%, and the precision was 10%. Under the same considerations, we present the metrics of the Silabo product (Table 10).

As shown in Table 10, the values of the metrics for the Silabo product are better than for the Estafeta product. This considering the sensitivity of the k parameter and the seed parameter in the clustering algorithm selected, for example, see the seventh and third version of the Silabo product.

As a general summary, Table 11 presents the average metrics for each software product with regards to

TABLE 10. Statistics for each version of the software product Silabo.

Metric	Mean	SD	Variance	Rank
Version 2				
TC Reduction rate (TR)	0.30	0.16	0.03	0.55
Recall	1	0	0	0
Precision	0.20	0.07	0.00	0.23
F-measure	0.33	0.09	0.01	0.29
Version 3				
TC Reduction rate	0.26	0.24	0.06	0.71
Recall	1	0	0	0
Precision	0.18	0.09	0.01	0.27
F-measure	0.29	0.12	0.01	0.35
Version 4				
TC Reduction rate	0.64	0.30	0.09	0.91
Recall	1	0	0	0
Precision	0.18	0.08	0.01	0.22
F-measure	0.29	0.13	0.02	0.32
Version 5				
TC Reduction rate	0.49	0.30	0.09	0.70
Recall	1	0	0	0
Precision	0.09	0.05	0.00	0.12
F-measure	0.16	0.08	0.01	0.20
Version 6				
TC Reduction rate	0.32	0.24	0.06	0.79
Recall	1	0	0	0
Precision	0.06	0.05	0.00	0.16
F-measure	0.12	0.08	0.01	0.27
Version 7				
TC Reduction rate	0.54	0.33	0.11	0.72
Recall	1	0	0	0
Precision	0.15	0.11	0.01	0.25
F-measure	0.25	0.16	0.03	0.37
Version 8				
TC Reduction rate	0.29	0.24	0.06	0.81
Recall	1	0	0	0
Precision	0.05	0.09	0.01	0.29
F-measure	0.09	0.14	0.02	0.43
Version 9				
TC Reduction rate	0.37	0.24	0.06	0.91
Recall	1	0	0	0
Precision	0.06	0.12	0.01	0.39
F-measure	0.09	0.17	0.03	0.54
Product Average				
TC Reduction rate	0.40	0.28	0.08	1.00
Recall	1	0	0	0
Precision	0.12	0.10	0.01	0.39
F-measure	0.20	0.15	0.02	0.54

the effectiveness. As shown in this table, we observe better results for the medium-sized software product (Silabo), with values of 40% 100%, 12% and 20% for the reduction rate of test cases, recall, precision and F-measure, respectively.

VI. DISCUSSION

Using the proposed approach, we obtained results that were analyzed using the following effectiveness metrics: suite reduction capability (TR), fault detection (R), precision (P) and F-measure. The metrics were collected for two software products (Estafeta and Silabo).

Regarding the average reduction capacity of the suite of test cases (TR), Table 9 list a reduction rate of 14% (0.14) for the Estafeta product. In the case of the Silabo product, the average reduction rate was 40% (0.40, see Table 10); in general, the results suggest that EM clustering for the proposed approach improves the reduction capacity of the suite when the number of test cases is increased.

TABLE 11. Metrics by software product.

Product	Estafeta (Mean)	Silabo (Mean)
TC Reduction rate	14%	40%
Recall	100%	100%
Precision	10%	12%
F-measure	17%	20%

On the other hand, consider the fault detection (Recall) metric: Tables 9 and 10 suggest that the proposed approach managed to capture 100% of the faults in the regression test; that is, R in the two case studies is of 100%, which had a favorable impact on the F-measure metric.

For the precision metric (P), according to Tables 9 and 10, on average, 12% was obtained for the Silabo product, while 10% was obtained for the Estafeta product, mainly due to the prioritization of the ability to capture all of the faults.

Finally, as regards to the F-measure metric, on average we observed 17% (0.17) for the Estafeta and 20% (0.20) for the Silabo product, which suggests improvement in the software regression tests under the clustering approach in medium-scale software products.

Regarding results of similar works, it can be noted that they presented considerable differences, owing basically to the regression testing approach used [14], [19], [20]. However, our results can serve as a reference for future similar RT proposals; also, our results tend to be more representative of a production setting.

A. STUDY LIMITATIONS

This section discusses the threats to validity considered in this work, for which the threats described in [56] were considered.

Internal threats: these are threats that can affect the validity of the results from the point of view of the execution of the empirical study. In this case, the design and generation of the test cases after the development of the software product were considered.

Regarding the clustering, a high sensitivity to the parameters k (number of clusters) and seed (initial centroid) of the Weka EM algorithm was evident; consequently, we decided not to set the parameter k. However, there are other works that address this issue, for example [54] and [55].

External threats: Although our results shed light on the use of the RT approach on a small- and a medium- size software product, it is necessary to perform studies with large-scale software products. To reduce this threat, the test cases were generated by the same developers, and the faults considered were real.

Threats to the construct: such threats are related to the metrics used in the empirical study. To reduce this threat, metrics related to the field of investigation were used. The metrics used for the assessment of the proposal are also used in other works related to the RT field.

VII. CONCLUSIONS

In this paper, we presented an approach for conducting incremental RT of database applications.

The proposed approach was validated empirically with two small- and medium-scale software products (Silabo and Estafeta), which can be considered as database software applications.

According to the results presented in Tables 9, 10 and 11, our proposed approach exhibits an improvement in the effectiveness of a regression test with respect to a complete regression test. In general, on average, our proposed approach reduced the number of test cases to 14% in product 1 (Estafeta) and 40% in product 2 (Silabo), both of them with a fault detection capacity of 100%, and on average with a precision of 10% and 12%, respectively.

This improvement is a result of the sensitivity of the clustering algorithms, due to their sensitivity to the number of clusters (k) and the initial random value (seed) of the process. It has a strong impact on the results of the clustering process. To reduce the impact of these parameters, we applied manipulations to the seed parameter and let the parameter k be random, obtaining on average values that yielded better results.

Based on our findings, it follows that combined analysis of unsupervised clustering techniques using unit testing techniques and DB tests can be applied in software regression tests, mainly those that are part of an iterative-incremental software development model.

To the best of our knowledge, this is the first work that considers a combination of the unit tests together with the schema of relational DBs and the use of an unsupervised clustering algorithm, in which a strong sensitivity of clustering algorithm parameters is emphasized, considering the dependence of the position (random values) in this type of clustering algorithm.

REFERENCES

[1] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw. Test. Verification Reliab.*, vol. 22, no. 2, pp. 67–120, Mar. 2012.

[2] R. H. Rosero, O. S. Gómez, and G. Rodríguez, "15 years of software regression testing techniques—A survey," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 5, pp. 675–689, Jun. 2016.

[3] G. Rothermel and M. J. Harrold, "A safe, efficient algorithm for regression test selection," in *Proc. Conf. Softw. Maintenance*, 1993, pp. 358–367.

[4] S. Parsa and A. Khalilian, "A bi-objective model inspired greedy algorithm for test suite minimization," in *Future Generation Information Technology*, Y. Lee, T. Kim, W. Fang, and D. Ślezak, Eds. Berlin, Germany: Springer, 2009, pp. 208–215.

[5] C. R. Panigrahi and R. Mall, "A hybrid regression test selection technique for object-oriented programs," *Int. J. Softw. Eng. Appl.*, vol. 6, no. 4, pp. 17–34, Oct. 2012.

[6] A. Pravin and S. Srinivasan, "Effective test case selection and prioritization in regression testing," *J. Comput. Sci.*, vol. 9, no. 5, pp. 654–659, May 2013.

[7] M. Kumar, A. Sharma, and R. Kumar, "Fuzzy entropy-based framework for multi-faceted test case classification and selection: An empirical study," *IET Softw.*, vol. 8, no. 3, pp. 103–112, Jun. 2014.

[8] H. K. N. Leung and L. White, "A cost model to compare regression test strategies," in *Proc. Conf. Softw. Maintenance*, 1991, pp. 201–208.

[9] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche, "Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system," *Softw. Test. Verification Reliab.*, vol. 25, no. 4, pp. 371–396, Jun. 2015.

[10] R. Mohanty, V. Ravi, and M. R. Patra, "The application of intelligent and soft-computing techniques to software engineering problems: A review," *Int. J. Inf. Decis. Sci.*, vol. 2, no. 3, pp. 233–272, 2010.

[11] R. O. Rogers, "Scaling continuous integration," in *Extreme Programming and Agile Processes in Software Engineering*, J. Eckstein and H. Baumeister, Eds. Springer, 2004, pp. 68–76.

[12] S. Biswas, R. Mall, M. Satpathy, and S. Sukumaran, "A model-based regression test selection approach for embedded applications," *SIGSOFT SoftwEng Notes*, vol. 34, no. 4, pp. 1–9, Jul. 2009.

[13] A. Podgurski and C. Yang, "Partition testing, stratified sampling, and cluster analysis," in *Proc. 1st ACM SIGSOFT Symp. Found. Softw. Eng.*, New York, NY, USA, 1993, pp. 169–181.

[14] S. Chen, Z. Chen, Z. Zhao, B. Xu, and Y. Feng, "Using semi-supervised clustering to improve regression test selection techniques," in *Proc. IEEE 4th Int. Conf. Softw. Test., Verification Validation (ICST)*, Mar. 2011, pp. 1–10.

[15] N. Gökce, F. Belli, M. Eminli, and B. T. Dinçer, "Model-based test case prioritization using cluster analysis: A soft-computing approach," *Turkey J. Elect. Eng. Comput. Sci.*, vol. 23, no. 3, p. 623, 2015.

[16] V. Vangala, J. Czerwonka, and P. Talluri, "Test case comparison and clustering using program profiles and static execution," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, New York, NY, USA, 2009, pp. 293–294.

[17] C. Zhang, Z. Chen, Z. Zhao, S. Yan, J. Zhang, and B. Xu, "An improved regression test selection technique by clustering execution profiles," in *Proc. 10th Int. Conf. Quality Softw. (QSIC)*, 2010, pp. 171–179.

[18] B. Guo, M. Subramaniam, and P. Chundi, "Analysis of test clusters for regression testing," in *Proc. IEEE 5th Int. Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2012, p. 736.

[19] Z. Chen, Y. Duan, Z. Zhao, B. Xu, and J. Qian, "Using program slicing to improve the efficiency and effectiveness of cluster test selection," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 21, no. 6, pp. 759–777, Sep. 2011.

[20] E. Rogstad and L. C. Briand, "Clustering deviations for black box regression testing of database applications," *IEEE Trans. Rehabil.*, vol. 65, no. 1, pp. 4–18, Mar. 2015.

[21] S. Yan, Z. Chen, Z. Zhao, C. Zhang, and Y. Zhou, "A dynamic test cluster sampling strategy by leveraging execution spectra information," in *Proc. 3rd Int. Conf. Softw. Test., Verification Validation (ICST)*, 2010, pp. 147–154.

[22] D. Zhang, Z.-H. Zhou, and S. Chen, "Semi-supervised dimensionality reduction," in *Proc. SIAM Int. Conf. Data Mining*, 2007, pp. 629–634.

[23] M. Bilenko, S. Basu, and R. J. Mooney, "Integrating constraints and metric learning in semi-supervised clustering," in *Proc. 21st Int. Conf. Mach. Learn.*, New York, NY, USA, 2004, p. 11.

[24] P. Berkhin, "A survey of clustering data mining techniques," in *Grouping Multidimensional Data*, J. Kogan, C. Nicholas, and M. Teboulle, Eds. Berlin, Germany: Springer, 2006, pp. 25–71.

[25] D. Willmor and S. M. Embury, "A safe regression test selection technique for database-driven applications," in *Proc. 21st IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2005, pp. 421–430.

[26] G. Rothermel and M. J. Harrold, "A safe, efficient regression test selection technique," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 2, pp. 173–210, 1997.

[27] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Inf. Softw. Technol.*, vol. 52, no. 1, pp. 14–30, 2010.

[28] Y.-F. Chen, D. S. Rosenblum, and K.-P. Vo, "TestTube: A system for selective regression testing," in *Proc. 16th Int. Conf. Softw. Eng.*, Los Alamitos, CA, USA, 1994, pp. 211–220.

[29] H. K. N. Leung and L. White, "A study of integration testing and software regression at the integration level," in *Proc. Conf. Softw. Maintenance*, 1990, pp. 290–301.

[30] P. E. Ammann and J. C. Knight, "Data diversity: An approach to software fault tolerance," *IEEE Trans. Comput.*, vol. C-37, no. 4, pp. 418–425, Apr. 1988.

[31] A. Orso, N. Shi, and M. J. Harrold, "Scaling regression testing to large software systems," in *Proc. 12th ACM SIGSOFT 12th Int. Symp. Found. Softw. Eng.*, New York, NY, USA, 2004, pp. 241–251.

[32] S. Kim, T. Zimmermann, E. J. Whitehead, Jr., and A. Zeller, "Predicting faults from cached history," in *Proc. 29th Int. Conf. Softw. Eng.*, Washington, DC, USA, 2007, pp. 489–498.
[33] G. Wikstrand, R. Feldt, J. K. Gorantla, W. Zhe, and C. White, "Dynamic regression test selection based on a file cache an industrial evaluation," in *Proc. Int. Conf. Softw. Test. Verification Validation*, 2009, pp. 299–302.
[34] M. Ficco, R. Pietrantuono, and S. Russo, "Bug localization in test-driven development," in *Proc. Adv. Soft Eng.*, 2011, pp. 2:1–2:18.
[35] J. Anderson, S. Salem, and H. Do, "Improving the effectiveness of test suite through mining historical data," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, New York, NY, USA, 2014, pp. 142–151.
[36] C. L. B. Maia, R. A. F. do Carmo, F. G. de Freitas, G. A. L. de Campos, and J. T. de Souza, "Automated test case prioritization with reactive GRASP," *Adv. Softw. Eng.*, vol. 2010, Jan. 2010, Art. no. 428521.
[37] W. Dickinson, D. Leon, and A. Podgurski, "Finding failures by cluster analysis of execution profiles," in *Proc. 23rd Int. Conf. Softw. Eng.*, Washington, DC, USA, 2001, pp. 339–348.
[38] A. Podgurski *et al.*, "Automated support for classifying software failure reports," in *Proc. 25th Int. Conf. Softw. Eng.*, Portland, OR, USA, 2003, pp. 465–475.
[39] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *Proc. 18th Int. Symp. Softw. Test. Anal.*, New York, NY, USA, 2009, pp. 201–212.
[40] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explorations Newslett.*, vol. 11, no. 1, pp. 10–18, 2009.
[41] E. G. Cartaxo, P. D. L. Machado, and F. G. O. Neto, "On the use of a similarity function for test case selection in the context of model-based testing," *Softw. Test. Verification Reliab.*, vol. 21, no. 2, pp. 75–100, Jun. 2011.
[42] R. A. Haraty, N. Mansour, and B. Daou, "Regression testing of database applications," in *Proc. ACM Symp. Appl. Comput.*, New York, NY, USA, 2001, pp. 285–289.
[43] K. Siau, *Advanced Topics in Database Research*. Calgary, AB, Canada: Idea Group, 2004.
[44] G. Rothermel and M. J. Harrold, "Analyzing regression test selection techniques," *IEEE Trans. Softw. Eng.*, vol. 22, no. 8, pp. 529–551, Aug. 1996.
[45] J. Tuya, J. Dolado, M. J. Suarez-Cabal, and C. de la Riva, "A controlled experiment on white-box database testing," *SIGSOFT Softw. Eng. Notes*, vol. 33, no. 1, pp. 8:1-8:6 2008.
[46] E. Rogstad, L. Briand, and R. Torkar, "Test case selection for black-box regression testing of database applications," *Inf. Softw. Technol.*, vol. 55, no. 10, pp. 1781–1795, Oct. 2013.
[47] E. Rogstad, L. Briand, R. Dalberg, M. Rynning, and E. Arisholm, "Industrial experiences with automated regression testing of a legacy database application," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2011, pp. 362–371.
[48] G. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*. Hoboken, NJ, USA: Wiley, 2007.
[49] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Mateo, CA, USA: Morgan Kaufmann, 2005.
[50] Y. G. Jung, M. S. Kang, and J. Heo, "Clustering performance comparison using K-means and expectation maximization algorithms," *Biotechnol. Biotechnol. Equip.*, vol. 28, pp. S44–S48, Nov. 2014.
[51] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong, "An empirical study of the effects of minimization on the fault detection capabilities of test suites," in *Proc. Int. Conf. Softw. Maintenance*, 1998, pp. 34–43.
[52] M. Fisher and S. Duskis, "Integration testing with JUnit," in *Spring Persistence—A Running Start*, Berkeley, CA, USA: Apress, 2009, pp. 153–162.
[53] S. Y. Kung, *Kernel Methods and Machine Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2014.
[54] G. Celeux and G. Govaert, "A classification EM algorithm for clustering and two stochastic versions," *Comput. Stat. Data Anal.*, vol. 14, no. 3, pp. 315–332, Oct. 1992.
[55] G. Hamerly and C. Elkan, "Alternatives to the K-means algorithm that find better clusterings," in *Proc. 11th Int. Conf. Inf. Knowl. Manage.*, New York, NY, USA, 2002, pp. 600–607.
[56] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Germany: Springer Science & Business Media, 2012.

RAÚL H. ROSERO received the Bachelor's degree in informatics engineering from the Central University of Ecuador, the Higher Diploma degree in software development process management from Army University, Ecuador, and the Master's degree in applied computer science from the Polytechnic School of Chimborazo. He is currently pursuing the Ph.D. degree in system engineering with the Universidad Nacional Mayor de San Marcos, Peru. His current research area is the verification and validation of software, testing software, and agile methods.

OMAR S. GÓMEZ received the Bachelor's degree in computer engineering from the University of Guadalajara, Mexico, the Master's degree in software engineering from the Center for Research in Mathematics, Mexico, and the Ph.D. degree in software and systems from the Technical University of Madrid, Spain. He has post-doctoral studies with the University of Oulu, Finland. He was a SENESCYT-Prometeo Researcher (initiative of the Ecuadorian government that seeks to strengthen research, academy, and knowledge transference) with the Technical School of Chimborazo, Ecuador. He is currently an Adjunct Associate Professor with the Technical School of Chimborazo. His research work focuses on experimentation in software engineering as well as on issues related to software quality, software verification, and software design. He was awarded with a membership to the Mexican network of researchers, where he currently holds the entry-level position of SNI-C from the Mexican National Council of Science and Technology.

GLEN RODRÍGUEZ (M'02) received the B.S. degree in system engineering from the Universidad Nacional de Ingenieria, Lima, Peru, in 1994, and the M.E. degree in information and computer science engineering and the Ph.D. degree in electronic and information engineering from the Toyohashi University of Technology, Toyohashi, Japan, in 2001 and 2004, respectively. Since 2006, he has been a Lecturer and later a Professor with the Universidad Nacional de Ingenieria. From 2008 to 2013, he was involved in the Cubesat project of this university responsible for the ground station. He teaches on the Graduate School of Universidad Nacional Mayor de San Marcos, Lima. His research interests are evolutionary algorithms, software testing, search-based software engineering, parallel processing, mobile communications, and information security.