Clustering Deviations for Black Box Regression Testing of Database Applications

Erik Rogstad and Lionel Briand *Fellow, IEEE*

***Abstract*—Regression tests often result in many deviations (differences between two system versions), either due to changes or regression faults. For the tester to analyze such deviations efficiently, it would be helpful to accurately group them, such that each group contains deviations representing one unique change or regression fault.**

Because it is unlikely that a general solution to the above problem can be found, we focus our work on a common type of software system: database applications. We investigate the use of clustering, based on database manipulations and test specifications (from test models), to group regression test deviations according to the faults or changes causing them. We also propose assessment criteria based on the concept of entropy to compare alternative clustering strategies.

To validate our approach, we ran a large scale industrial case study, and our results show that our clustering approach can indeed serve as an accurate strategy for grouping regression test deviations. Among the four test campaigns assessed, deviations were clustered perfectly for two of them, while for the other two, the clusters were all homogenous. Our analysis suggests that this approach can significantly reduce the effort spent by testers in analyzing regression test deviations, increase their level of confidence, and therefore make regression testing more scalable.

***Index Terms*—Software regression testing, test analysis, clustering, regression test deviations.**

Abbreviations and Acronyms

SOFIE	The Norwegian Tax Accounting System, the case study system
DART	DAtabase Regression Testing, a tool developed for regression testing of database applications
CTE-XL	Classification Tree Editor - eXtended Logics, a commercial tool for classification tree modeling.
SQL	Structured Query Language
PL/SQL	Procedural Language or Structured Query Language
DML	Data Manipulation Language
EM	Expectation Maximization, an algorithm used for clustering in our case
LSA	Latent Semantic Analysis
SIR	Software-artifact Infrastructure Repository

Notations

$\mathbf{D}$	The set of deviations
$\mathbf{C}$	The set of clusters
$\mathbf{c_j}$	A cluster j in C
$\mathbf{d_i}$	Deviation type i in D
$\mathbf{d_{ij}}$	Deviation type i in cluster j
$\mathbf{E_D}$	Deviation entropy
$\mathbf{E_C}$	Cluster entropy

I. Introduction

REGRESSION testing is a highly important but time consuming activity [1]. A great deal of work has been performed on devising and evaluating techniques for selecting, minimizing, and prioritizing regression test cases [2]. Such techniques are necessary, but unfortunately not sufficient to help scale regression testing to large, complex systems. Indeed, in practice, even with efficient prioritization or selection, numerous regression test deviations may need to be analyzed to determine if they are due to a regression fault or simply the effect of a change. A problem that has been largely ignored so far, but which is highly important in practice, is how to cope with the many discrepancies (deviations) that can be observed when running regression test cases on a new version of a system. In other words, how can we help testers analyze such deviations, which can be due to either changes or regression faults, and decide what are their actual causes. While this is an important problem for regression testing of all types of systems, the types of output from regression tests and their deviations are logically dependent on the type of system being tested. Thus, we do not believe there is a general solution to assist in the analysis of regression test deviations for all types of systems. In this case study, we focus on database applications, where the regression test output consists of deviations in database manipulations. Although our proposed solution may not be limited to database applications, this is the context where we have evaluated it, and where the reported results are most likely to generalize and be accurate.

In the context of regression testing of database applications, we have proposed a black-box regression test methodology and tool [3]. We model the input domain of the system under test as classification trees, using the tool Classification Tree Editor - eXtended Logics (CTE-XL) [4], which are used to generate abstract test cases, and in turn executable test cases. During the execution of the system under test, we automatically observe and log database manipulations. We use the database manipulation logs from the original version of the system under test as the test oracle. Deviations between the original system output and that of the modified version are considered a potential regression fault, and the tester determines whether or not they actually are by inspecting the details of deviation logs. The number of deviations from a regression test campaign will vary with the scope of the test, but our experience is that, in many cases, this number can be expected to be large and exceed the inspection capacity of the testers. However, past test campaigns show that the number of distinct deviations,

E. Rogstad works for the Department of Software Engineering at Simula Research Laboratory, 1325 Lysaker, Norway. E-mail: erikrog@simula.no

L. Briand works for the SnT Centre at the University of Luxembourg, L-2721 Luxembourg, Luxembourg. E-mail: lionel.briand@uni.lu

that is groups of deviations corresponding to the same fault or change, is far lower than the number of deviations itself. Thus, this paper is about finding an accurate strategy for grouping regression test deviations that are due to the same change or regression fault, for the tester to work through the deviations far more efficiently, while still finding most regression faults.

Our goal is to identify patterns among deviations by making use of clustering algorithms, the ultimate objective being to identify groups of deviations caused by identical changes or regression faults. As elaborated in Section V when investigating the research literature within the context of testing, clustering has most widely been adopted in test case optimization techniques, such as test case selection and prioritization, while the problem we address has been little explored. We perform a thorough assessment of which type of data leads to the most accurate clustering results, using information collected about database manipulations and test case properties. One important contribution of this paper is to assess the accuracy of clustering for grouping regression test deviations in an industrial context, with real changes and regression faults, and evaluate the practical impact for the tester. Our regression test approach targets database applications in general, because it is based on information that should be easy to collect for most systems falling into that category.

The results show that clustering indeed can serve as an accurate strategy for grouping regression test deviations according to their cause, being changes or regression faults. For two out of the four test campaigns evaluated in the study, we achieved perfect clustering results with the clustering strategy that fared best, meaning that there were as many clusters as distinct changes and regression faults, and each cluster only contained one type of deviation (homogenous). For the other two test campaigns, all clusters were still homogenous, but the numbers of clusters were slightly larger than the numbers of distinct deviations, i.e. deviations caused by the same fault or change were spread across more than one homogeneous cluster. That level of accuracy, when grouping deviations, is expected to significantly reduce the test analysis effort, and thus improves the scalability of any regression testing strategy.

The remainder of the paper is organized as follows. Section II describes the challenges addressed by this paper, and the industrial context of our study, which in many ways are representative of environments where large database applications are being developed. Section III outlines our proposed solution and the specifics of the clustering strategy, whereas Section IV describes the industrial case study and reports empirical results. Related works are presented in Section V, and the paper is concluded in Section VI.

II. CONTEXT AND BACKGROUND

The main purpose of this section is to provide information about the context in which the problem was identified and defined. This problem, however, is of a general nature, and likely to be present in most database applications. Providing a concrete description of the background of our industrial case study will help the reader understand the scope, importance, and generality of the problem.

Petersen and Wohlin [5] proposed a checklist for context documentation in industrial software engineering. We cover elements of the product facet, the process facet, and the practices and tools and techniques facet in this section. As the subject system is a customized application for the Norwegian Tax Department, we do not regard the organization and market facet relevant in this case, and will not elaborate on that any further. The details of how the case study was performed and the people involved is described in Section IV-B.

A. Setting

The Norwegian Tax Department maintains the Norwegian tax accounting system (SOFIE), a system whose main purpose is to collect tax from all taxpayers in Norway. The system was developed as a customized application for the Norwegian Tax Department during the late 2000s, and entered its maintenance phase in 2011. SOFIE serves the daily operation of more than 3,000 end users (i.e. taxation officers), and handles annual tax revenues of approximately 600 billion Norwegian Kroner. It is therefore important to preserve system quality upon changes and new releases, to avoid additional effort to end users and expensive mistakes both for the taxpayers and administration. The system is a very large database application, built on Oracle database technology, developed with standard Structured Query Language (SQL) and Procedural Language or Structured Query Language (PL/SQL) programming, along with Oracle Forms user interfaces. In terms of size, the system has more than 1,000 tables in its main database schema, and contains more than 700 PL/SQL packages, and a variety of other database application artifacts (triggers, functions, procedures, views, etc.), in total constituting more than 2.5 million lines of code. The core of the system is batch driven, with periodically scheduled batch jobs processing centralized tasks for all Norwegian taxpayers, whereas the end users access the application through a web interface. The batch jobs process large amounts of data, and their complexity stems from the wide range and diversity of possible input data to process. Though it is vital for the tax department to avoid releasing defects into the core of the system, the system is complex and difficult to evolve to respond to changes in the tax laws, and is thus prone to regression faults. Therefore, an appropriate regression test procedure is required to guarantee the quality of SOFIE. The releases of SOFIE follow a typical waterfall development process, and the regression testing referred to here is mainly targeted towards system level testing.

B. Regression test procedure

Because manual regression testing is not deemed to scale for large database applications, we proposed a partially automated regression test procedure tailored to database applications [3]. It compares executions of a changed version of the program against the original version of the program, and identifies deviations, which are differences in the way the database is manipulated between the two executions. In each test execution, the database manipulations are logged according to a specification by the tester indicating the tables and columns to

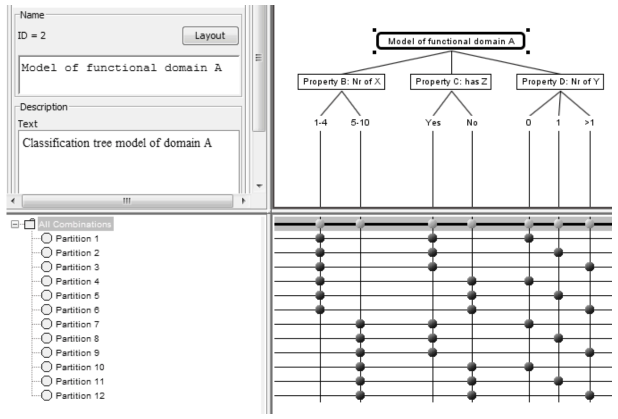


Fig. 1. An example of a classification tree model in CTE-XL, and the generated partitions (combinations of equivalence classes) that form abstract test cases.

monitor (hereby called *test configuration*). More specifically, we log

Data Manipulation Language (DML) statements, namely inserts, updates, and deletes (whether they are prepared or parameterized statements or not) per test case. These database manipulations are captured using dynamically generated database triggers, based on the test configuration, during the execution of the test. As detailed in Rogstad et al. [3], the database manipulations from each execution are logged in a structured way, and are compared across system versions to produce a set of deviations, indicating either a correct change or a regression fault. The comparison is performed using SQL set operations (minus and union) to capture the differences in database manipulations between two test runs, grouped per test case. In the context of SOFIE and any tax system in general, a test case matches a taxpayer, and all logged data are grouped per taxpayer. The output of regression testing will therefore indicate whether or not there was a deviation between two system versions in the tax calculation for a particular taxpayer. Additionally, to make test selection more systematic and cost effective, following common industry practice, we model the input domain as a classification tree, like the example shown in Fig. 1. All relevant properties of the system are modeled as classifications (e.g., Property B: Nr of X), and split into equivalence classes (e.g., 1 through 4, and 5 through 10) representing the range of possibilities in the inputs of the system. For the sake of generality, as other input modeling techniques could have been used, we hereby refer to the classifications in the model as *model properties*, and the values of the equivalence classes (leaf classes in the model) as *model property values*. Based on the model, we generate input partitions (combinations of equivalence classes) according to a given coverage criteria (i.e. pair-wise, three-wise, and all combinations), and each partition represents an abstract test case.

Such a test procedure has the benefit of pinpointing exactly which test case results have changed between system versions. As an example, let us consider a batch processing 10,000 tax calculations from which 100 deviations are identified. We can conclude that no regression faults were visible in the 9,900 test cases with no deviations, while our attention can be directed towards the 100 tax calculations with deviations. Though automated deviation analysis helps a great deal, the tester still has to inspect deviations to determine their cause, and this is a manual, time consuming activity. The number of deviations typically varies with the size of the test suite and the scope of the changes in the particular release under test. But in most cases, the number of deviations is expected to be large, and will require prioritization given the usual project time constraints. Throughout previous SOFIE releases, we have experienced a varying number of deviations across regression test campaigns. This experience is shown in Table I, which shows the number of test cases, the total number of deviations, and the number of distinct groups of deviations with the same cause for various test campaigns.

C. Problem formulation

The number of deviations vary from 8 to 522, and though eight deviations are clearly manageable for a tester to manually inspect, in most campaigns there are many more. Imagine spending 15 minutes on average inspecting each deviation, which is a very conservative number in our experience; then analyzing 522 deviations would require 130 hours. Though available time for a tester to carry out a test campaign is a function of its size and importance, spending 130 hours on inspecting deviations is rarely a possibility. We need a solution to dramatically reduce the time spent in such inspections. By introducing systematic methods for selecting test cases for execution [6], we have been able to reduce the size of the test suites while retaining a good coverage of test scenarios. However, the number of deviations resulting from regression test campaigns is still, in most cases, much larger than what a tester can realistically handle. A trade-off is therefore required, and the tester needs assistance to determine which subset of deviations to focus on to identify as many different regression faults as possible.

Any deviation between the original system output or observable behavior and that of the modified version is seen as a potential regression fault. When presented with the list of deviations, the tester needs support to systematically process them, as relying on experience alone is rather uncertain in such a context where a large number of deviations, mostly different from what was observed in previous versions, have to be analyzed. Without any automated decision support, what we observed is that the current practice for a tester is basically to start at the top of the arbitrary ordered list of deviations and work through as far as time budget permits. But if there is not enough time to inspect all of them, the tester is then left with great uncertainty. Our experience shows that there always is redundancy among the deviations in the sense that several of them are caused by the same change or fault. As shown in Table I, the number of deviations is always larger than the number of distinct deviations, that is the number of groups of deviations related to distinct changes or faults. For a tester to avoid spending unnecessary effort on analysis, we need a systematic way to group the deviations, so that ideally each group matches one distinct deviation only. Then the tester

TABLE I
A SUBSET OF HISTORIC DATA ON THE NUMBER OF DEVIATIONS PRODUCED FROM REGRESSION TESTS

Test campaign	# Test cases	# Deviations	# Distinct deviations
1	711	33	7
2	3144	182	11
3	5670	522	15
4	1570	8	2
5	94	48	2
6	560	47	2
7	560	84	2
8	151	43	3

TABLE II
EXAMPLE OF THE OUTPUT OF A REGRESSION TEST, I.E. THE DETAILS OF THE DEVIATIONS

Test case	Table	Column	Old value	New value	Program version
1	A	X	5,000	0	Baseline
1	A	X	5,000	4,000	Delta
2	A	X	Inserted	2,500	Baseline
2	A	Y	Inserted	TAX	Baseline
3	A	X	350	Deleted	Delta
3	A	Y	INTEREST	Deleted	Delta

would only have to inspect one deviation from each group to complete the analysis. Given a perfect grouping strategy, the tester would be left inspecting 15 deviations as opposed to 522, or 2 as opposed to 84, to take a couple of examples from the data shown in Table I. The potential savings are therefore highly significant.

Thus, the problem we try to address is to find a strategy to accurately group deviations resulting from the same change or regression fault, in order for the tester to inspect as few deviations as possible while still remaining confident of finding all regression faults.

III. CLUSTERING REGRESSION TEST DEVIATIONS

Recall that, in our context, a deviation points out a difference between the original program and the changed version of the program for a particular test case. For each test case resulting in a deviation, a detailed list of the specifics of the deviation is provided by our tool. That is, a deviation for a particular test case will indicate the concrete database fields that deviate between the two system versions. Table II gives a concrete example of how the details of the deviations may look like in a trivial case. In this case, three test cases (1, 2, and 3) resulted in a deviation. In test case 1, column X in database table A was updated from 5,000 to 0 in the baseline run of the program, whereas it was updated to 4,000 in the delta run. For test case 2, a record was inserted in database table A with the values 2,500 for column X, and TAX for column Y in the baseline run, but not in the delta run. For test case 3, a record was deleted containing the values 350 in column X, and INTEREST in column Y in the delta run, but not in the baseline run. Given a large number of deviations, each composed of a great deal of detailed information, the amount of information to process is large. In a typical regression test, there may be thousands of rows of deviation details. As a result, recognizing patterns across test cases, indicating similarities among deviations, is a highly difficult task.

We have chosen to base our solution for grouping deviations on clustering, that is algorithms which aim at discovering groups and structures in data in such a way as to maximize the distance between groups, and minimize the distance within groups [7]. In our specific context, we would like to identify a clustering approach that could help us cluster the deviations as accurately as possible, the goals being to have 1) all deviations resulting from the same regression fault or change within the same cluster, and even more importantly, 2) homogenous clusters containing only deviations related to the same fault or change. In other words, the ideal output would have as many clusters as there are regression faults and changes, and each cluster would only contain deviations matching a unique fault or change.

The most important decision to make is to decide what data to feed into the *clustering algorithm*. As mentioned in Section II, we model the input domain of the system under test as a classification tree, from which we generate abstract test cases. In addition, during the execution of the system under test, the regression test tool automatically captures all database manipulations executed by each test case in the format <*table, column, operation, old value, new value*>. Thus, the information available as input for clustering is twofold: 1) which model property values each test case covers; and 2) the deviation output describing the specifics of the deviation at a table, column, operation, and value level. It is difficult to tell, a priori, which type of input data will lead to the most accurate grouping. Assessing different input sources and their combinations is therefore an objective of our empirical investigation in the next section.

The input data used for clustering is encoded in a binary matrix, as shown in the upper part of Fig. 2. The matrix contains one row per deviation, and for each deviation we indicate with a Boolean value whether a particular table, column, operation, or model property value characterizes the deviation. For example, $table_{1..a}$ denotes each potential table in the deviation output (retrieved from the test configuration), which will be marked 1 (true) or 0 (false) depending on whether or not the table is present in the deviation output for a particular deviation. Similarly, $column_{1..b}$ denotes all monitored database table columns, and $Operation_{1..c}$ denotes the type of database operation that caused the deviation, which is either an insert, update, or delete. This structure is applied at a table level, stating whether the deviation was due to an insert, update, or delete applied to a table. *Model property value*$_1..d$ denotes whether or not the test case causing the deviation covers a certain model property value or not, where a model property value is a leaf class from the classification tree model (e.g., 1 through 4 for Property B in Fig. 1).

To exemplify, let us generate an input matrix to the clustering based on the example model shown in Fig. 1, and the deviation output shown in Table II. Let *Partition 1* from the example model be the test case specification of *Test case 1* in the deviation output. Similarly, *Partition 2* specifies *Test case 2*, and *Partition 3* specifies *Test case 3*. In this small example, the only entries in the test configuration (tables and columns to monitor) is column X and Y in database table A. Based on the deviation output given in Table II, and the

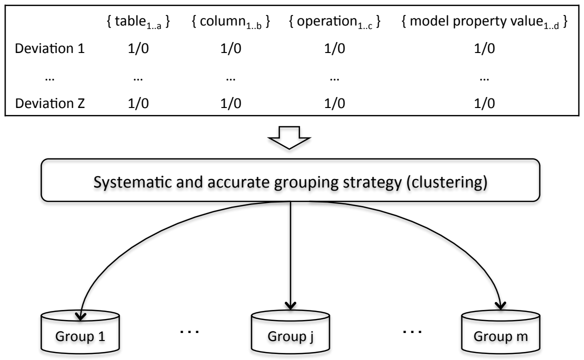


Fig. 2. The grouping strategy; encode available information and input to a clustering algorithm to group deviations

model property values given by Fig. 1, the encoded binary matrix would then look like the one shown in Table III. The data from the binary matrix, either representing one single input source (tables, columns, database operations, or model property values) or their combinations, will be used as input to the clustering algorithm.

Once test cases are executed, and deviations are clustered based on the data presented above, we rely on a deviation analysis strategy that is aimed at minimizing the number of deviations to analyze, while covering all regression faults or changes. It is assumed that the regression test analyst will analyze, in turn, at least one deviation from each deviation cluster, in a random order. If each cluster captures deviations corresponding to a unique cause, i.e. change or regression fault, this result will satisfy our objectives. Specific accuracy measures for our clustering algorithm will be discussed in the next section.

Many clustering techniques require the number of clusters to be determined beforehand. However, in our case, we do not know the target number of clusters up front, as the number of distinct deviations will vary across regression test campaigns. Thus, an important requirement when choosing a cluster algorithm to serve our purpose is to select one that does not require a predefined number of clusters as input. However, we would like to limit the maximum number of clusters, as historical data show that the number of distinct deviations, i.e. groups of deviations corresponding to distinct causes, typically represent 2% to 7% of the total number of deviations. Being able to set an upper bound for the number of clusters to approximately 10% of the number of deviations would prevent us from having an unrealistically large number of clusters. Because a part of the deviation analysis strategy is to inspect at least one deviation from each cluster, we target as few clusters as possible. Clustering, as we defined it, can be applied to any type of test case. But of course, if we were doing a white-box approach, we might use coverage information as well for doing clustering.

For practical reasons, we chose to consider cluster algorithms provided by Weka [7], as it is well documented, has an active wiki and discussion forum, and provides an open source programmable API which is easy to integrate into our existing solution. Among the clustering algorithms provided by Weka, the probabilistic algorithm Expectation Maximization (EM) is the only one that does not require the number of clusters to be predefined (it determines the number of clusters to create based on cross-validation of the input data), and that does provide the ability to set an upper bound for the number of clusters. Consequently, the EM clustering algorithm was our preferred choice for conducting deviation grouping. Inputs to the algorithm include the data set to cluster, the acceptable error to converge, the maximum number of iterations, and optionally the maximum number of clusters. In the case of this study, we have used 1.0E-6 as the minimum allowable standard deviation for convergence, while the maximum number of iterations is set to 100 [7]. Additionally, as stated above, based on historical data, we set the maximum number of clusters to 10% of the number of deviations. If the maximum number of clusters is not given a priori, EM will resort to cross-validation on the input data to set the number of clusters.

In our context, we have a set of vectors representing the deviations, and a set of clusters capturing groups of deviations with common causes. EM will start with an initial guess for clustering, based on the cross-validation of the input data. Then, it estimates an initial distribution of deviations across clusters, i.e. the probability distribution for each deviation, indicating its probability of belonging to each of the clusters. Through iterations, the algorithm will try new parameters based on the outcome of the previous iteration to maximize the log-likelihood of the deviations across the clusters. The algorithm finishes when the distribution parameters converge, or when reaching the maximum number of iterations. The Weka implementation of EM clustering does not use a distance function, but rather estimates a mixture model for clustering, by taking a naive Bayes approach which models each attribute *s*-independent of the others given the cluster. We refer to McLachlan and Krishnan [8] for further details about the EM clustering algorithm.

IV. Case study

This section outlines the case study according to the guidelines given by Runeson and Höst [9], and presents the results of the study.

A. Objective and research questions

The objective of the case study is to group regression test deviations in such a way that deviations resulting from the same regression fault or change are likely to be contained within the same group, and so that groups only contain deviations related to the same fault or change. This grouping is expected to increase the efficiency of the tester when performing regression test analysis as, if the above objectives are reached, only one deviation from each group needs to be inspected, rather than picking deviations at random within a limited time frame or inspecting all of them.

The following research questions have been derived from the overall objective of the case study.

RQ1 Can clustering serve as an automated, accurate grouping strategy for grouping regression test deviations?

TABLE III
A SMALL, ARTIFICIAL EXAMPLE OF A BINARY MATRIX USED AS INPUT TO CLUSTERING.

Deviation	**Table**	**Column**		**Operation**			**Model Property Values**						
	A	A.X	A.Y	A.INSERT	A.UPDATE	A.DELETE	B.1-4	B.5-10	C.YES	C.NO	D.0	D.1	D.>1
1	1	1	0	0	1	0	1	0	1	0	1	0	0
2	1	1	1	1	0	0	1	0	1	0	0	1	0
3	1	1	1	0	0	1	1	0	1	0	0	0	1

TABLE IV
THE SUBJECT REGRESSION TEST CAMPAIGNS FOR THE CASE STUDY

Test Campaign	# Test cases	# Deviations	# Distinct deviations [Frequency]
1	94	48	2 [41 (fault), 7 (fault)]
2	560	47	2 [19 (fault), 28 (change)]
3	560	84	2 [56 (change), 28 (fault)]
4	151	43	3 [3 (fault), 10 (change), 30 (change)]

RQ2 What kind of input data to the clustering process yields the most accurate grouping of the deviations?

RQ3 How much effort can the tester be expected to save in regression test and deviation analysis, when systematically grouping test deviations prior to analysis, as opposed to the current random inspection?

B. The case and data collection

We investigated test deviations from the regression tests of SOFIE, the tax accounting system of the Norwegian Tax Department. Most of the context documentation is given in Section II, whereas we detail the specifics of how the case study was conducted here.

The test team of SOFIE ran four regression test campaigns using the regression test tool DAtabase Regression Testing (DART) [3] on consecutive releases of SOFIE. One of their testers was responsible for the test execution, while one tester (having the necessary domain knowledge) conducted the analysis of the test results. As expected, the specifics of the tests varied across releases, as different parts of the system were affected by the scope of the release. Once the deviations were classified as either regression faults or changes to be verified, we applied our clustering strategy, and evaluated its accuracy and potential efficiency gains for the testers. Details about the particular test campaigns used in the study are provided in Table IV, showing the number of test cases executed, and the number of deviations resulting from the test. As for the number of tables and columns monitored during the regression test execution, the order of magnitude for the test campaigns in our case study was 10-12 tables, spanning 63-83 table columns, whereas the size of the classification tree models varied from 26 to 127 model property values.

For the purpose of the case study, we collected data about actual regression tests across various releases of SOFIE. More specifically, information was collected about the test configuration, the model of the test domain, and the list of deviations produced, which were manually analyzed and categorized by the test team as regression faults or changes. According to the definitions given by Runeson and Höst [9], this process corresponds to second degree data collection, as we are directly in control of the collection of our raw data through observations of the regression tests on the subject system.

The SOFIE project was selected as it is large, complex, and representative of database applications developed by the tax department and other public administrations. Furthermore, and it is important when studying regression testing, SOFIE undergoes frequent changes due to general maintenance and changes in taxation laws.

C. Evaluation

The accuracy of the clustering algorithm, which is in question in both RQ1 and RQ2, is measured in terms of its entropy. Entropy is a measure of unpredictability [10], and we apply it to evaluate two distinct dimensions: deviation, and cluster.

- Deviation entropy is the measure of the spread across clusters of deviations matching the same fault or change. Ideally all related deviations should only be contained in one cluster.
- Cluster entropy is the measure of the purity of a cluster. Ideally a cluster should only contain deviations matching one specific fault or change.

Entropy gives us a normalized measure of the purity of each cluster, and the spread of each type of deviation across clusters, while additionally giving an overall measure for a set of clusters, enabling us to compare the overall clustering accuracy for different kinds of inputs. A perfect clustering would have as many clusters as there are regression faults and changes, and each cluster would only contain deviations matching a unique fault or change, thus yielding an entropy of zero. The worse the grouping, the higher the entropy level, with the worst case occurring when deviations matching specific faults or changes are equally spread across the clusters. In the remainder of the text, we will use deviation group to mean an actual group of deviations matching a specific fault or change. The formula for measuring deviation entropy (E_D) [10], more specifically the spread of a particular deviation group d_i in D across the m clusters (C), is given by (1). For each of the m clusters c in C, we compute the number of deviations of type i that belong to cluster c_j (d_{ij}) divided by the total number of deviations in group i ($|d_i|$). This ratio is then employed in the usual information theory entropy formula for all clusters and groups. In general, in all our entropy calculations, we have used the natural logarithm ($log_e(p)$), and taken $0log(0)$ to be 0, to be consistent with the limit $\lim_{p\to0+} plog(p) = 0$, stating that when p converges sufficiently close to zero, $plog(p)$ evaluates to zero.

$$\mathrm{E_D}(d_i, C) = -\sum_{j=1}^{m}(\frac{d_{ij}}{|d_i|})log(\frac{d_{ij}}{|d_i|}) \tag{1}$$

The formula for measuring cluster entropy (E_C), namely the purity of a cluster c_j, is given by (2). For each of the n deviation groups, we compute the number of occurrences of d_i in c_j (d_{ij}) divided by the total number of deviations in c_j. This ratio is then employed again using the information theory entropy formula for all clusters and groups.

$$\mathrm{E_C}(D, c_j) = -\sum_{i=1}^{n}(\frac{d_{ij}}{|c_j|})log(\frac{d_{ij}}{|c_j|}) \tag{2}$$

The formula for total deviation entropy (E_{D_TOT}), and total cluster entropy (E_{C_TOT}) are given in (3), and (4), respectively. They are basically the sum of (1) and (2), across all deviation groups, and clusters, respectively.

To demonstrate the calculations in practice, we will calculate the entropy for one deviation and one cluster from the examples given in Fig. 3. The calculation of deviation entropy for deviation d_1 in Fig. 3(a) is illustrated in (5). In total, there are 5 deviations of type d_1, meaning the cardinality of deviation d_1 ($|d_1|$) is 5. The deviation group d_1 is spread across two clusters, c_1 and c_4, containing 2 and 3 deviations of type d_1, respectively. The other two clusters do not contain any deviations from group d_1, so those parts of the equation evaluate to zero. The calculation of cluster entropy for cluster c_2 in Fig. 3(b) is illustrated in (6). The cardinality of cluster c_2 ($|c_2|$) is 8, and it contains 6 deviations of type d_2, and 2 deviations of type d_3. As cluster c_2 does not contain any deviations from deviation group d_1, the first part of the equation evaluates to zero. All other deviation and cluster entropy calculations for the examples given in Fig. 3 would evaluate to zero.

$$\mathrm{E_{D\_TOT}}(D, C) = -\sum_{i=1}^{n}\mathrm{E_D}(d_i, C) \tag{3}$$

$$\mathrm{E_{C\_TOT}}(D, C) = -\sum_{j=1}^{m}\mathrm{E_C}(D, c_j) \tag{4}$$

$$\mathrm{E_D}(d_1, C) = -\sum_{j=1}^{4}(\frac{d_{1j}}{|d_1|})log(\frac{d_{1j}}{|d_1|}) = (\frac{2}{5})log(\frac{2}{5}) + (\frac{0}{5})log(\frac{0}{5}) + (\frac{0}{5})log(\frac{0}{5}) + (\frac{3}{5})log(\frac{3}{5}) \approx 0.6730 \tag{5}$$

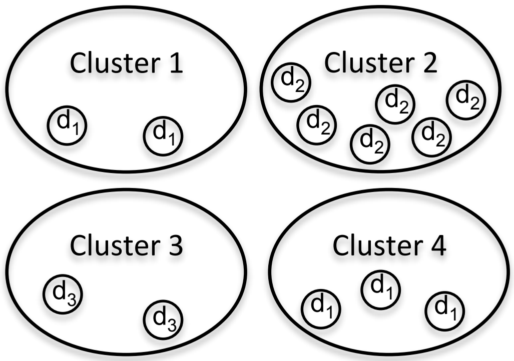
$$\mathrm{E_C}(D, c_2) = -\sum_{i=1}^{3}(\frac{d_{i2}}{|c_2|})log(\frac{d_{i2}}{|c_2|)} = (\frac{0}{8})log(\frac{0}{8}) + (\frac{6}{8})log(\frac{6}{8}) + (\frac{2}{8})log(\frac{2}{8}) \approx 0.5623 \tag{6}$$

From a practical viewpoint, it is more important to obtain zero cluster entropy than zero deviation entropy. If the cluster entropy evaluates to zero, we are certain that each cluster only matches one regression fault or change. Under such circumstances, the tester would always conduct a complete deviation analysis when inspecting one deviation from each group, only with the potential risk of having to analyze a few more deviations than necessary if the deviation entropy is not perfect. On the contrary, if the cluster entropy significantly differs from zero, then inspecting only one deviation from each group would represent a significant risk. The ideal target is a total entropy level of zero; but if we have to compromise, a zero cluster entropy is preferred over a zero deviation entropy. That is, the situation in Fig. 3(a) is preferred over the situation in Fig. 3(b).

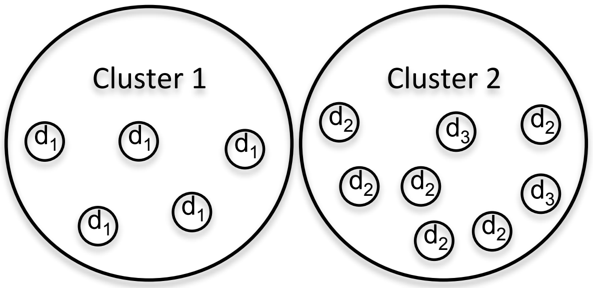
To evaluate the effort spent by the tester in analyzing regression test deviations for RQ3, we measure the expected number of deviations to be inspected to identify all deviation groups. This approach relies on an assumption regarding how the clustering results are used, and in our context we assume a round robin random selection of deviations from the clusters. That is, the tester would randomly select clusters in a round robin manner, and within each cluster select a random deviation (under the assumption that they all match the same regression fault or change). For each test campaign, we count the number of deviations to inspect until all deviation groups are covered. We repeat the exercise 1000 times to account for randomness in sampling, and report basic statistic measures such as minimum, median, maximum, average, and standard deviation regarding the number of deviations. Additionally, we conduct non-parametric, two-tailed Mann-Whitney U-Tests [11] to check the statistical significance of the difference in inspection effort when the cluster-based approach was compared against the random inspection currently in use. We also report $\hat{A}_{12}$ effect size measurements to assess the practical significance of the differences. An $\hat{A}_{12}$ effect size measurement value of 0.5 indicates that there is no difference between the two samples compared, while the further away from 0.5, the larger the effect size between the samples. We have categorized the effect size into the standard Small, Medium, and Large categories, which are usually defined as Small $<$ 0.10, 0.10 $<$ Medium $<$ 0.17, and Large $>$ 0.17, the value being the distance from the 0.5 threshold [12].

D. Results

This section reports our case study results when clustering is applied to group regression test deviations. During the course of the study, four regression test campaigns were executed, from which data were collected. Only a subset of the eight test campaigns given as examples in Table I were included in the current study as, due to changes in the project environment, not all required data were available for some past releases. Relevant data for the different regression test suites are shown in Table IV. For each of them, we went through the following



(a) Clustering with zero cluster entropy (all clusters are pure). 13 deviations of three different types d_1, d_2, and d_3 spread across four clusters.



(b) Clustering with zero deviation entropy (all deviations of the same type are contained within the same cluster). 13 deviations of three different types d_1, d_2, and d_3 spread across two clusters.

Fig. 3. Zero cluster entropy versus zero deviation entropy.

activities.

1) Preparing input data: For each type of input data (tables, columns, database operations, and model property values), we generated an input matrix (as shown in Fig. 2) for the *clustering algorithm*. In addition, we generated matrices combining all combinations of the input types.
2) Measuring entropy: For each of the various input matrices provided, we carried out the clustering, as described in Section III, and measured the entropy level for each deviation group, and each generated cluster, to assess clustering accuracy. It should be mentioned that the execution time of the clustering is negligible (0 to 2 seconds), and is therefore not given any further attention in the study.
3) Assessing expected inspection effort: Based on the most accurate clustering approach overall, we estimated the expected number of deviations that must be inspected by the tester to inspect all distinct deviations. This estimate is compared against an inspection strategy where deviations are selected at random, because this is the current practice when inspecting the regression test deviations for our subject system, and probably in many other test environments.

1) Accuracy of clustering: Table V shows a summary of the *deviation entropy* and the *cluster entropy* per cluster input for each of the subject regression test campaigns. The full list of details, regarding entropy per deviation and cluster, along with the actual distribution of deviations across clusters per input type, and per test, is made available in the Appendix.

The first noteworthy result is that for three of the test campaigns (two, three, and four), a perfect clustering was achieved for at least one type of cluster input, i.e. each row in Table V. By perfect, we mean a total entropy level of zero per test campaign, indicating that there are as many clusters as the number of distinct regression faults and changes, and all clusters are homogenous. A concrete example is the input pair *columns and operations* for test campaign number two. Another interesting point is that four types of inputs result in a zero cluster entropy for all test campaigns, which is our priority. The input types are *tables*, *tables and operations*, *columns and operations*, and *tables and columns and operations*. As stated in Section IV-C, achieving a zero cluster entropy is most important from a practical point of view, as it ensures that all cluster groups are homogeneous, containing instances of one deviation group only. Such a situation allows the tester, in all confidence, to conduct the deviation analysis by only inspecting one deviation from each group. The fact that a total entropy level of zero was obtained for three out of four test campaigns, along with the fact that zero cluster entropy was achieved for all test campaigns for certain input types, shows that we are able to cluster regression test deviations with a high level of accuracy. Thus, regarding RQ1, we conclude that clustering can serve as an automated, accurate strategy for grouping regression test deviations.

2) Preferred input data: The best cluster input overall is the combination of *columns and operations*. This strategy resulted in perfect clustering for two of the test campaigns, while also ensuring zero cluster entropy for the other two test campaigns. Using only tables as input yielded similar results, but the overall entropy is nevertheless lower when using the combination of columns and operations. Relying only on tables when clustering would also be more sensitive to the number of tables monitored during the test. If very few tables were monitored, the approach would most likely be too coarse and inaccurate. Thus, relying on the combination of columns and operations seems to be at an appropriate level of granularity. In general, using one source of input seems too limited for the clustering algorithm to yield the best results. For example, using only columns or operations separately provides unsatisfactory results, while combining them leads to very accurate clustering. On the other hand, using all input type combinations or even triples seems unnecessary. With regards to RQ2, we conclude that the combination of *columns and operations* is the input combination yielding the most accurate grouping of the regression test deviations.

Please recall from Section III that in our approach we have used a binary representation of our regression test database manipulations and test case properties. Alternatively, it would also have been possible to count the number of occurrences of each table, column, and operation in the deviation output. If we were to venture into even more details, one could also account for the order of operations; however, that would be slightly more intricate to encode into meaningful cluster inputs. Nevertheless, because the results using only the presence of each database element are already so accurate, there was not much

TABLE V
ENTROPY MEASUREMENTS FOR EACH COMBINATION OF INPUT VALUES, AND FOR EACH OF THE FOUR REGRESSION TEST CAMPAIGNS.

			Entropy							
			Test 1		**Test 2**		**Test 3**		**Test 4**	
			Deviation	Cluster	Deviation	Cluster	Deviation	Cluster	Deviation	Cluster
Cluster input	Single	Tables	0.827	0.000	1.129	0.000	0.000	0.000	0.000	0.000
		Columns	0.920	1.310	0.616	0.474	0.458	0.000	1.295	0.540
		Operations	0.720	0.690	0.410	0.462	0.000	0.000	0.563	0.000
		Model Property Values	0.526	0.685	1.385	0.679	0.000	0.637	1.619	0.855
	Pair	Tables and columns	0.826	0.410	0.693	0.000	0.458	0.000	1.089	0.000
		Tables and operations	0.416	0.000	1.428	0.000	0.000	0.000	0.943	0.000
		Tables and model property values	1.722	0.410	2.277	1.235	0.000	0.000	0.000	0.540
		Columns and operations	**0.416**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**1.089**	**0.000**
		Columns and model property values	0.979	0.690	0.683	0.000	1.040	0.000	1.074	0.540
		Operations and model property values	1.242	1.140	1.779	0.637	0.693	0.000	0.000	0.540
	Three	Tables and columns and operations	0.813	0.000	1.428	0.000	0.000	0.000	1.089	0.000
		Tables and columns and model property values	1.062	0.000	1.684	0.000	0.000	0.000	0.673	0.540
		Tables and operations and model property values	0.494	0.000	0.598	0.000	1.974	0.271	0.991	0.540
		Columns and operations and model property values	1.015	0.689	1.684	0.000	1.890	0.000	0.000	0.540
	All	Tables and columns and operations and model property values	0.416	0.000	1.747	0.683	0.693	0.000	1.089	0.540

room for improvement in our case study. Such alternatives may, however, be worth investigating in other contexts.

One last thing to note is that nearly all clustering results based on model properties yield poor results, indicating they are not suited to cluster deviations. A plausible explanation is that the model describes the input domain of the regression test rather than the output of the test. Although the two are related, the relationship is complex, and the results suggest that the information available in the deviation output is more relevant to clustering the deviations.

3) Inspection effort: Given accurate deviation clusters, we assume here an inspection strategy which involves picking a random cluster in a round robin manner, and then, within each cluster, picking a random deviation. This procedure is continued until all faults and changes are covered, and we measure, as a surrogate measure for effort, the expected number of deviations a tester would need to inspect to cover all deviation groups. This procedure is repeated 1000 times to account for randomness in sampling. The results are shown in Table VI, where *Clustered Round Robin* is compared against the *Random Inspection* strategy currently in use in the SOFIE project.

Compared to inspecting all deviations, the effort of the tester is dramatically reduced when clustering deviations and inspecting test results following the round robin procedure stated above. In three out of four regression test campaigns, the median number of deviations to inspect to cover all faults and changes corresponds to the actual number of different faults and changes. In other words, we obtain perfect results. In the worst case, we need to inspect a few more deviations, but never more than the number of formed clusters. And recall from Section III that, as part of the clustering strategy, we impose an upper bound for the number of clusters to 10% of the number of deviations. Thus, the number of clusters remains manageable, and the number of deviations to inspect remains very reasonable.

The statistical tests and the $\hat{A}_{12}$ effect size measurements are provided in Table VII, where the p-value from the statistical tests are reported, along with the categorized $\hat{A}_{12}$ measures with the actual value given in parentheses. The results of the tests show that the cluster-based inspection is significantly better than random inspection for all test campaigns, and the difference in effect size is large for all campaigns, according to standard evaluation scales. This result is also clearly visible from the differences in median and average values in Table VI. Between the cluster-based and random inspections, the relative increase in the number of deviations to inspect is between approximately 50% (test campaigns 2, and 3 with 2 vs. 3 deviations) and 150% (test campaigns 1, and 4 with 2 vs. 5, and 4 vs. 10 deviations, respectively). When addressing RQ3, because we expect the difference in effort spent on analyzing deviations to be proportional to these differences, our results suggest significant savings when using clustering.

Nevertheless, random inspection yielded better results than we expected (the median number of inspections is 5, 3, 3, and 10). This result stems from a limited number of deviation groups (distinct deviations) present in our subject regression test campaigns. In test campaigns 2 and 3, both deviation groups present are represented by a large number of deviations, which favor a random strategy. For test campaigns 1 and 4, where there exist deviation groups containing relatively few deviations, and in the latter case also more deviation groups, the random approach shows a significant increase in the average number of deviations to inspect and its standard deviation. Therefore, good results for a random inspection strategy are less likely in releases with a larger number of regression faults and changes, especially in the presence of an uneven distribution of corresponding deviations. A detailed analysis shows that the results from the random approach are always worse than those of the clustering approach, and show a significantly higher standard deviation (ranging from 1.568 to 7.451 as opposed to 0.000 to 0.831), thus resulting in more uncertainty in terms of the number of inspections one may have to go through. The worst maximum number for random inspections is 38, which is a significant portion of the 43 deviations present in the test campaign. In contrast, the cluster-based inspection approach provides far more predictable results, which is important in our context to prevent

TABLE VI
THE NUMBER OF DEVIATIONS NEEDED TO BE INSPECTED BY THE TESTER TO COVER ALL DISTINCT DEVIATIONS

Test		Inspection strategy	Minimum	Median	Maximum	Average	Std.-dev.
Test Campaign 1	Deviations: 48	Clustered Round Robin	2	2	3	2.324	0.468
	Distinct deviations: 2	Random Inspection	2	5	24	6.564	4.745
Test Campaign 2	Deviations: 47	Clustered Round Robin	2	2	2	2	0.000
	Distinct deviations: 2	Random Inspection	2	3	12	3.111	1.568
Test Campaign 3	Deviations: 84	Clustered Round Robin	2	2	2	2	0.000
	Distinct deviations: 2	Random Inspection	2	3	13	3.337	1.828
Test Campaign 4	Deviations: 43	Clustered Round Robin	3	4	5	4.097	0.831
	Distinct deviations: 3	Random Inspection	3	10	38	12.063	7.451

TABLE VII
MANN-WHITNEY U-TESTS, AND $\hat{A}_{12}$ EFFECT SIZE MEASUREMENTS WHEN COMPARING INSPECTION EFFORT ACROSS CLUSTERED INSPECTION, AND RANDOM INSPECTION

Test campaign	Cluster-based (A), Random (B)		
	P-value	Superior	Effect size
1	< 0.0001	A	Large (0.817)
2	< 0.0001	A	Large (0.754)
3	< 0.0001	A	Large (0.772)
4	< 0.0001	A	Large (0.888)

regression faults from slipping though. From our results, which are of course to be confirmed in other studies, this systematic approach guarantees that at least one deviation is analyzed for each fault or change by inspecting only one deviation from each cluster.

To summarize, in practice when conducting a cluster-based inspection, the tester would analyze one deviation from each cluster (one-per-cluster sampling). And most importantly, given the results we obtained, the tester would do that analysis while remaining confident he or she will cover all deviation groups, which are distinct faults and changes. In our case study, that approach would correspond to the following results for the four test campaigns. The tester would inspect 3 out of 48, 2 out of 47, 2 out of 84, and 5 out of 43 deviations for the four test campaigns, respectively. This inspection would definitely entail highly significant savings. In contrast, when inspecting an arbitrarily ordered list of deviations, the tester would have to inspect all of them, or as many of them as possible given time constraints, to gain sufficient confidence in the test results. In the latter case, the tester would have no guarantee that he has covered all or most faults and changes. Thus, a cluster-based inspection strategy is in practice far better than a random inspection, and yields significant savings, especially in the context of large regression test suites.

E. Threats to Validity

In this section, we discuss the generality and potential threats to validity of the research, following the classification scheme of Yin [13], as further described for software engineering in the guidelines by Runeson and Höst [9].

1) Construct validity: We needed to measure the accuracy of clustering when applied for grouping regression test deviations. By measuring entropy across two dimensions, namely deviation entropy and cluster entropy, we capture two complementary aspects of accuracy required to investigate RQ1 and RQ2. Other accuracy measures, such as standard recall and precision for classifiers, were also considered to assess our clustering results. But unlike classification, clustering results are not compared to a ground truth, telling us what should be the correct classification. We could of course have measured precision and recall based on the majority of changes and faults in each cluster, but that would have yielded specific values for each cluster, and not overall measurement values for all clusters. That is why we chose to rely on entropy, which gives us, for a clustering algorithm and a set of inputs, a normalized measure of the purity of the clusters, and the spread of the deviation groups across clusters.

The inspection effort, addressed in RQ3, is expected to vary across test campaigns and deviations. Thus, we have used the expected number of deviations to inspect as a surrogate measure of inspection effort, and reported standard descriptive statistics, such as median, average, and standard deviation. Inspection effort, on average and over many deviations, is expected to be proportional to the number of inspected deviations.

2) Internal validity: To assess the accuracy of deviation clustering, we relied on the manual inspection and categorization of the regression test deviations performed by software engineers. The resulting categories of regression test deviations served as a baseline against which we compared deviation clusters. Ideally, more than one person should have taken part in this process to avoid mistakes. However, because of the cost associated with the task of inspecting all the deviations from each test campaign in a real development environment, this task was performed by one engineer only. The manual inspection was conducted physically independent of and prior to the clustering approach, and given the experience and competence of the tester, we expect mistakes in classifying deviations to be unlikely, and we do not see how this would bias our results towards higher clustering accuracy.

The use of classification trees to specify and automate test cases is not a limitation of the applicability of our approach as clustering results clearly show that model property values, the only information specific to classification trees, are not a useful clustering input. Second, the use of classification trees for black-box testing is widespread and similar to other black-box test specification techniques.

3) External validity: The fact that the study only involves one system is a potential threat to the external validity of the study. However, the focus on a representative database application, using standard and widely used database technology, combined with real regression test campaigns and faults, helps mitigate this threat. Ideally, one would always want to run multiple studies to better ensure the generalizability of the results. However, to replicate the study, we need actual regression test suites, for actual versions of the system, with known and actual regression faults diagnosed by engineers. Then we need, for each version, to re-run all test cases, and monitor the manipulations on the database, the latter being properly populated with data. That type of data is just not available for open source systems, and to replicate our study on another real world, large scale system would be a major undertaking requiring the collaboration of another development organization.

Regarding the representativeness of our subject system, recall from Section II that the subject system is built on standard Oracle database technology. As Oracle is the world market leader both in terms of applications platforms and database management systems [14], the subject system is built on technology that is widespread across the world. Moreover, the chosen subject system is very similar to many other such database applications developed by the Norwegian government and other administrations, with a typically long lifespan. These systems process large amounts of data, leading to a wide variety of possible test scenarios, which makes testing challenging, and the need for assistance crucial in the analysis and diagnosis of regression tests. We have also evaluated several test campaigns, spanning different test models and different functional domains of the system under test, to ensure a certain variety within our case study. The information used as input to clustering should also generalize in the context of database applications, as this kind of information should be easy to collect from any system falling into that category. The database information is also captured through a mechanism separate from the code running on top of the database, i.e. whether the manipulations are performed by a batch, a procedure, or a graphical user interface does not matter. Captured will be all manipulations performed by any code executed by a test case in the executed test suite on the table columns monitored during test execution.

Borg and Runeson [15] showed a significant dependency between the results obtained and the data characteristics of the studied case in the context of information retrieval in software engineering. Transferred to our case, this could mean that the calibration of the clustering method, that is selecting which input data provides the best clustering output, may depend on the context. If so, it would require a full deviation analysis to calibrate the method in a new context. Then, the gain in inspection effort would not materialize until the method is satisfactorily calibrated, which would be at best after the first test run. Note, however, that the best clustering approach in our case study showed stable, consistent results across the test campaigns. Nevertheless, over time, the need to recalibrate the method at certain intervals may arise. But given a system with a long lifespan, numerous regression tests would be required throughout its existence, thus making the benefits of clustering deviations significant across many releases.

4) Conclusion Validity: In Section IV-D3, we executed the deviation sampling 1000 times per test campaign and per sampling strategy to account for randomness. We used a non-parametric statistical test to compare statistically independent samples, corresponding to different strategies, that make no assumption about data distributions. Thus, we have sufficient statistical power, and use appropriate tests to draw statistical conclusions.

V. Related work

This case study is about supporting the analysis of regression test discrepancies by clustering them to guide their inspection. Thus, we have chosen to place our work in the context of clustering related to testing and fault categorization. We refer to Rogstad et al. [3] for works related to regression testing of database applications, which are not directly relevant to what is presented in this paper as we do not make any assumption about how test cases are selected.

In the context of testing, clustering has been most widely used for test optimization purposes, such as test case selection and test case prioritization. Test case selection aims at selecting a subset of all test cases in a test suite for execution, while test case prioritization allocates a priority to each test case resulting in a test suite of ordered test cases. Both techniques usually try to maximize fault detection, either in the selected subset or in the high priority test cases. Early clustering work in relation to these topics emerged in the last 20 years, but most particularly surged in the last 4 to 5 years. Although most of the papers are focused on test cases, there are also a few papers applying clustering to fault classification. The following subsections summarize the main contributions of clustering in each application area. We will remain brief on the papers related to test optimization techniques, while going more in depth for the ones on failure clustering, as those are more closely related to our work.

A. *Clustering-based test case selection*

Dickinson et al. [16] evaluated the effectiveness of cluster analysis for finding failures, by building on the work of Podgurski et al. [17], [18] on what was labeled as *Cluster Filtering*. Cluster filtering is the combination of a clustering metric and a sample strategy used to filter executions (test cases) to select those exhibiting unusual behavior, hypothesizing this approach would lead to early fault detection. They combine varying inputs to the cluster algorithm with different sampling strategies to conclude that cluster filtering is more effective (detects more faults) than simple random sampling, the best approach being adaptive sampling combined with input data giving extra weight to unusual profile features.

To select a subset of test cases for regression testing, Parsa et al. [19] cluster test cases based on their execution profile, and then sample test cases until the code coverage level of the original test suite is satisfied. Compared to the well known H-algorithm [20], their approach leads to improved fault detection rates for similar sized test suites, or greater test suite reduction for similar fault detection rates.

Focusing on the elimination of duplicate test cases, Vangala et al. [21] address test case minimization in their clustering-based selection approach. They use the program profile, including code coverage, the number of times a block or arc was executed, and static analysis of the source code as a basis for clustering, using thresholds for redundancy detection. On average, they identified 10-20% redundant test cases with a 70% accuracy.

Zhang et al. [22] proposed a regression test selection technique by clustering execution profiles of modification-traversing test cases. Compared to the safe selection technique proposed by Rothermel et al. [23], their approach produced smaller test suites, retaining most fault-revealing test cases. Chen et al. [24] extended the work by Zhang et al., and introduced semi-supervised clustering into their regression test selection approach. They used two pair-wise constraints between test cases, namely must-be-in-same-cluster, and cannot-be-in-same-cluster, to guide the clustering process. The constraints are derived from the previous test results, and used to state whether pairs of test cases must be in the same cluster, or not. The results showed improved fault detection rates up to 30% when compared to unsupervised clustering.

As opposed to the above-mentioned works, which all are white-box clustering approaches based on execution profiles at the source code level, Sapna et al. [25] applied clustering for specification-based testing. To achieve effective test case selection, they clustered test scenarios from UML activity diagrams to select dissimilar test cases. While the results showed improved fault detection rates over random selection, results did not lead to the desired property of being able to detect most faults with small numbers of test cases.

In the context of Web Applications, Liu et al. [26] collected user-sessions from the application, which in turn were used as test cases. They clustered user-sessions to select them for testing. With the reduced set of test cases, they achieved nearly the same fault detection rate and code coverage level as with the entire test suite.

Wang et al. [27] investigated test case selection in observation-based testing. They also clustered test cases based on their execution profile, but then assigned weights to the clustered test cases based on a ranking metric. They calculated a score for every function covered by the test case based on complexity, where functions receiving high scores are considered more likely to contain bugs. Compared to an execution-spectra-based sampling strategy [28], the weighted approach showed improved fault detection rates with reduced test suite sizes.

To summarize the cluster-based approaches to selection, most of them 1) take a white-box approach by using information from execution profiles at a source code level as a basis for the clustering, Sapna et al. being the exception with a black-box strategy based on activity diagrams; 2) use Agglomerative Hierarchical Clustering or k-means clustering; and 3) are evaluated on one or more open source programs.

B. Clustering-based test case prioritization

To benefit from human expert knowledge in test case prioritization, Yoo et al. [29] employ clustering to scale the use of manual pair-wise comparisons of test cases to large test suites. By clustering the test cases hierarchically, the workload of pair-wise comparisons is divided into smaller, more manageable tasks, called intra-cluster and inter-cluster prioritization. Compared to classic statement-coverage based ordering, their clustering-based prioritization fared better for 9 out of 16 test suites, with an average improvement in the fault detection rate of 6.5%.

Simons et al. [30] used clustering for the purpose of regression test case prioritization. Test cases are clustered based on characteristics from the test profiles and execution behavior, followed by an initial random one-per-cluster sampling. If they come across a failed test case (based on information from the previous test), the k nearest neighbors from that test case are selected from within the cluster. Their approach labeled *adaptive failure pursuit sampling* was compared to original failure pursuit sampling, yielding a significant increase in the average fault detection rate.

Contrasting with most existing works, Carlson et al. [31] conducted an industrial case study on the use of clustering to improve test case prioritization. They cluster test cases based on code coverage and test case information from the version control system, and then prioritize test cases within and between clusters by using software metrics such as code complexity, fault history, and code coverage. Prioritization incorporating clustering did on average provide a 40% improvement in the fault detection rate over prioritization without clustering.

Arafeen et al. [32] investigate whether requirements information, when added to traditional code analysis information, would improve cluster-based test case prioritization. They use text mining to cluster test cases based on requirement similarities and code complexity information, combined with different sampling strategies. Their results indicate that the use of requirement information during test case prioritization can be beneficial, showing an average improvement in the fault detection rate of approximately 30% compared to code metric prioritization.

To summarize, the clustered-based approaches to prioritization all 1) take a white-box approach by using information from execution profiles at the source code level as a basis for the clustering, 2) use Agglomerative Hierarchical Clustering or k-means clustering, and 3) are evaluated on one or more open source programs from SIR, Carlson et al. being the exception reporting on an industrial case study.

C. Clustering for failures classification

Whereas the works mentioned in the previous sections focus on test case clustering, the works presented in this section address the classification of failures using clustering to work through them more efficiently. Therefore, the work presented in this section is the most similar to our work, as the focus is to find out which failures to investigate.

Many software products today have the ability to detect some of their own runtime failures, and automatically generate failure reports to facilitate debugging. Whereas automated failure reporting is a significant asset, it is likely to produce more reports than the developers can effectively handle. Podgurski et al. [33] use clustering to group failure reports which exhibit similar execution profiles with respect to a set of selected

TABLE VIII
OVERVIEW OF STUDIES RELATED TO FAILURE CLUSTERING

Reference	Task	Input information	Context	Subject programs
[33]	Cluster failure reports	Automatically generated failure reports	White-box testing	3 open source programs (compilers)
[34]	Cluster failed test cases	Test case execution trace	White-box GUI testing	A messenger application
[35]	Cluster failed test cases	Terms from the executed source code	White-box testing	1 open source program from SIR

features. They used the Clara clustering algorithm from S-PLUS based on the k-medoids clustering criterion [36], while utilizing the Calinski-Harabasz index [37] to determine the number of clusters. When evaluated on three open source subjects (all compilers), they found that, in most of the clusters created, a majority of the failures were due to the same cause. Overall, groups of failures with the same cause tended to form fairly cohesive clusters. However, a few large, non-homogeneous clusters were created containing sub-clusters consisting of failures with the same cause.

When a large number of failed test cases are reported from a test automation system, recommending some representative test cases as a starting point for debugging can be helpful to developers. In the context of graphical user interface testing, Chien-Hsin et al. [34] proposed clustering as a solution to provide such help. More concretely, they collected test case execution traces through instrumented code, calculated a similarity matrix according to an adapted version of the Needleman-Wunsch Algorithm [38], and clustered the failed test cases with Agglomerative Hierarchical Clustering. The developer was then expected to fix failed test cases following a one-per-cluster sampling strategy. A case study on a messenger application (55,000 lines of code) with seeded faults indicated reduced bug fixing effort when following the proposed strategy.

While many of the clustering-based approaches in testing rely on control-flow analysis, mostly considering the execution profiles of the test cases, DiGiuseppe et al. [35] suggest utilizing latent-semantic analysis (LSA) to capture the semantic intent of an execution. They use LSA in a concept-based execution clustering technique to categorize executions that fail due to the same fault, similar to what Arafeen et al. [32] subsequently proposed. The main difference with the latter is that it is applied at a code execution level rather than based on requirements information. They instrument and execute the program, parse the source code to identify words, compute the weighted term frequency and inverse document frequency for each term, and then cluster the test cases using agglomerative hierarchical clustering. When evaluated on one open source program from the Software-artifact Infrastructure Repository (SIR), concept-based clustering was able to achieve similar cluster purity (94%), but substantially reduced the number of clusters (by 70%) when compared to pure control-flow-based clustering. Indeed, this technique was able to detect similar executions despite differing control paths being executed.

A summary of the works most closely related to ours is provided in Table VIII. To summarize, in contrast to those works, the current paper is an industrial case study focusing on clustering regression test deviations based on test case specifications and runtime database changes in the context of database applications. No existing work has addressed that important problem. In other words, our work differs with respect to the task supported, the type of information used as a basis for clustering, the types of test cases, and the application context.

VI. CONCLUSION

One main challenge in large scale regression testing is to analyze observed deviations on new software versions to decide whether they are symptoms of regression faults or the logical result of changes. In a specific context, we have investigated the use of clustering to group regression test deviations to assist and prioritize their analysis. Our focus is on the functional, black-box regression testing of database applications, which are widespread in many application domains, and usually require substantial regression testing over a long lifespan. We compare runtime database manipulations between executions of the original and changed versions of the system under test. We collect information such as the tables and columns subject to manipulation, along with the old and new values of the fields. Note that such information should be general, and easy to collect for any database application. The output of the regression test is the set of test case deviations, along with the specific details of the deviations. The analysis of past regression test campaigns shows that many deviations are caused by the same regression fault or change. Our study is about determining whether clustering could serve as an accurate strategy for grouping regression test deviations according to the faults or changes that caused them. We based our clustering strategy on the EM clustering algorithm because it does not require the number of clusters to be determined beforehand, but provides the opportunity to set an upper limit for it. Such an algorithm is fed with information regarding various aspects of the database elements being manipulated, and properties of the test cases exhibiting deviations, which is then used to compute distances between deviations.

We conducted a case study, in a real development setting, based on a large, critical database application developed by the Norwegian Tax Department. Four concrete regression test campaigns, covering three different parts of the system under test, were executed and analyzed by the test engineers in the project. That is, they executed the regression tests, and inspected all deviations from the test campaigns, while categorizing them into regression faults or changes. We then applied our proposed clustering strategy on different combinations of input data, and evaluated the accuracy of the deviation grouping for each type of input combination to determine what information was relevant for our objective. The results show that, for three out of the four test campaigns, clustering accuracy was perfect for at least one type of input. Additionally, four types of input yielded homogenous clusters (containing only one type of deviations) for all test campaigns. Homogenous clusters are our main priority as they imply that a tester inspecting one deviation from each cluster (one-per-cluster sampling) is certain to cover all regression faults and changes causing deviations. Overall, the information that led to the most accurate deviation clustering was the combination of columns and operations in the deviation output, indicating differences in columns and operations being manipulated across versions for a given test case. That specific combination of input resulted in perfect accuracy for two of the test campaigns, while achieving perfectly homogeneous clusters for the two others, with a limited degree of dispersion across more than one cluster for common-cause deviations. In terms of inspection effort, for the four test campaigns under study, and when using a one-per-cluster sampling strategy, it was estimated that testers would only have to analyze a very small percentage of deviations, specifically three out of 48, two out of 47, two out of 84, and five out of 43, respectively, while still covering all regression faults and changes. This change is a dramatic increase in efficiency for deviation analysis in regression testing, and a way to achieve much higher confidence when analyzing deviations under time constraints. Such results are more specifically important for scaling regression testing to large database applications.

A possible future addition could be to help the tester understand the relation between model properties and the clusters. For instance, we could investigate whether it is possible to infer relations such as that all deviations in *Cluster A* relate to the same equivalence class for one or more model properties, and are the only ones in the test suites to do so. That would strongly indicate the types of properties that cause different types of deviations, which would further help the tester or developer when looking for the cause of regression test failure.

APPENDIX
ENTROPY MEASUREMENT DETAILS

This appendix contains the details of the clustering process per regression test, namely how the deviations are distributed across clusters, the deviation entropy per deviation type, and the cluster entropy per cluster.

- Table IX shows entropy details for test campaign 1.
- Table X shows entropy details for test campaign 2.
- Table XI shows entropy details for test campaign 3.
- Table XII shows entropy details for test campaign 4.

TABLE IX
Entropy details for Test 1: For each type of input the deviation entropy is evaluated per type of deviation and the cluster entropy is evaluated per cluster

				Deviation Entropy				Cluster Entropy										Total Entropy		
		Test 1	Number of clusters	Fault 49166		Fault 49167		Cluster 0		Cluster 1		Cluster 2		Cluster 3		Cluster 4		Total Fault Entropy	Total Cluster Entropy	Total Entropy
				Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy			
Cluster input	Independent	Tables	5	(30,6,0,4,1)	-0,827	(0,0,7,0,0)	0,000	(30,0)	0,000	(6,0)	0,000	(0,7)	0,000	(4,0)	0,000	(1,0)	0,000	-0,827	0,000	-0,827
		Columns	3	(4,35,2)	-0,509	(6,0,1)	-0,410	(4,6)	-0,673	(35,0)	0,000	(2,1)	-0,637					-0,920	-1,310	-2,229
		Operations	3	(4,6,31)	-0,720	(0,7,0)	0,000	(4,0)	0,000	(6,7)	-0,690	(31,0)	0,000					-0,720	-0,690	-1,410
		Model Property Values	2	(9,32)	-0,526	(7,0)	0,000	(9,7)	-0,685	(32,0)	0,000							-0,526	-0,685	-1,212
	Pair	Tables and Columns	3	(35,6,0)	-0,416	(0,1,6)	-0,41	(35,0)	0,000	(6,1)	-0,41	(0,6)	0					-0,826	-0,410	-1,237
		Tables and Operations	3	(35,6,0)	-0,416	(0,0,7)	0	(35,0)	0,000	(6,0)	0	(0,7)	0					-0,416	0,000	-0,416
		Tables and Model Property Values	5	(6,9,9,0,17)	-1,312	(1,0,0,6,0)	-0,410	(6,1)	-0,410	(9,0)	0,000	(9,0)	0,000	(0,6)	0,000	(17,0)	0,000	-1,722	-0,410	-2,132
		Columns and Operations	3	(35,6,0)	-0,416	(0,0,7)	0,000	(35,0)	0,000	(6,0)	0,000	(0,7)	0,000					-0,416	0,000	-0,416
		Columns and Model Property Values	3	(6,13,22)	-0,979	(7,0,0)	0,000	(6,7)	-0,690	(13,0)	0,000	(22,0)	0,000					-0,979	-0,690	-1,670
		Operations and Model Property Values	4	(5,1,30,5)	-0,832	(1,0,0,6)	-0,410	(5,1)	-0,451	(1,0)	0,000	(30,0)	0,000	(5,6)	-0,689			-1,242	-1,140	-2,382
	Triple	Tables, Columns and Operations	5	(31,4,0,4,2)	-0,813	(0,0,7,0,0)	0,000	(31,0)	0,000	(4,0)	0,000	(0,7)	0,000	(4,0)	0,000	(2,0)	0,000	-0,813	0,000	-0,813
		Tables, Columns and Model Property Values	5	(2,5,10,0,24)	-1,062	(0,0,0,7,0)	0,000	(2,0)	0,000	(5,0)	0,000	(10,0)	0,000	(0,7)	0,000	(24,0)	0,000	-1,062	0,000	-1,062
		Tables, Operations and Model Property Values	3	(33,0,8)	-0,494	(0,7,0)	0	(33,0)	0,000	(0,7)	0	(8,0)	0					-0,494	0,000	-0,494
		Columns, Operations and Model Property Values	3	(35,0,6)	-0,416	(0,2,5)	-0,598	(35,0)	0,000	(0,2)	0,000	(6,5)	-0,689					-1,015	-0,689	-1,704
	All	Tables, Columns, Operations and Model Property Values	3	(6,35,0)	-0,416	(0,0,7)	0,000	(6,0)	0,000	(35,0)	0,000	(0,7)	0,000					-0,416	0,000	-0,416

TABLE X
Entropy details for Test 2: For each type of input the deviation entropy is evaluated per type of deviation and the cluster entropy is evaluated per cluster

		Test 2	Number of clusters	Fault 49451		Change 49446		Cluster 0		Cluster 1		Cluster 2		Cluster 3		Cluster 4		Total Deviation Entropy	Total Cluster Entropy	Total Entropy
				Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy			
Cluster input	Independent	Tables	4	(0,16,0,3)	-0,436	(14,0,14,0)	-0,693	(0,14)	0,000	(16,0)	0,000	(0,14)	0,000	(3,0)	0,000			-1,129	0,000	-1,129
		Columns	3	(0,18,1)	-0,206	(24,4,0)	-0,410	(0,24)	0,000	(18,4)	-0,474	(1,0)	0,000					-0,616	-0,474	-1,090
		Operations	2	(0,19)	0,000	(24,4)	-0,410	(0,24)	0,000	(19,4)	-0,462							-0,410	-0,462	-0,872
		Model Property Values	3	(0,9,10)	-0,692	(14,0,14)	-0,693	(0,14)	0,000	(9,0)	0,000	(10,14)	-0,679					-1,385	-0,679	-2,064
	Pair	Tables and Columns	3	(0,19,0)	0,000	(14,0,14)	-0,693	(0,14)	0	(19,0)	0	(0,14)	0,000					-0,693	0,000	-0,693
		Tables and Operations	5	(0,0,16,0,3)	-0,436	(10,4,0,14,0)	-0,992	(0,10)	0,000	(0,4)	0	(16,0)	0	(0,14)	0	(3,0)	0	-1,428	0,000	-1,428
		Tables and Model Property Values	5	(4,2,6,0,7)	-1,297	(6,6,0,16,0)	-0,980	(4,6)	-0,673	(2,6)	-0,562	(6,0)	0,000	(0,16)	0,000	(7,0)	0,000	-2,277	-1,235	-3,512
		Columns and Operations	2	(19,0)	0,000	(0,28)	0,000	(19,0)	0,000	(0,28)	0,000							0,000	0,000	0,000
		Columns and Model Property Values	3	(19,0,0)	0,000	(0,16,12)	-0,683	(19,0)	0,000	(0,16)	0,000	(0,12)	0,000					-0,683	0,000	-0,683
		Operations and Model Property Values	4	(6,6,0,7)	-1,096	(12,0,16,0)	-0,683	(6,12)	-0,637	(6,0)	0,000	(0,16)	0,000	(7,0)	0,000			-1,779	-0,637	-2,415
	Triple	Tables, Columns and Operations	5	(0,0,16,0,3)	-0,436	(10,4,0,14,0)	-0,992	(0,10)	0,000	(0,4)	0,000	(16,0)	0,000	(0,14)	0,000	(3,0)	0,000	-1,428	0,000	-1,428
		Tables, Columns and Model Property Values	5	(0,0,10,0,9)	-0,692	(10,4,0,14,0)	-0,992	(0,10)	0,000	(0,4)	0,000	(10,0)	0,000	(0,14)	0,000	(9,0)	0,000	-1,684	0,000	-1,684
		Tables, Operations and Model Property Values	3	(19,0,0)	0,000	(0,20,8)	-0,598	(19,0)	0,000	(0,20)	0,000	(0,8)	0,000					-0,598	0,000	-0,598
		Columns, Operations and Model Property Values	5	(0,0,10,0,9)	-0,692	(10,4,0,14,0)	-0,992	(0,10)	0,000	(0,4)	0,000	(10,0)	0,000	(0,14)	0,000	(9,0)	0,000	-1,684	0,000	-1,684
	All	Tables, Columns, Operations and Model Property Values	5	(0,0,16,0,3)	-0,436	(11,5,0,8,4)	-1,311	(0,11)	0,000	(0,5)	0,000	(16,0)	0,000	(0,8)	0	(3,4)	-0,683	-1,747	-0,683	-2,430

TABLE XI
ENTROPY DETAILS FOR TEST 3: FOR EACH TYPE OF INPUT THE DEVIATION ENTROPY IS EVALUATED PER TYPE OF DEVIATION AND THE CLUSTER ENTROPY IS EVALUATED PER CLUSTER

		Test 3	Number of clusters	Change 45024		Fault 50000		Cluster 0		Cluster 1		Cluster 2		Cluster 3		Cluster 4		Cluster 5		Total Deviation Entropy	Total Cluster Entropy	Total Entropy
				Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy			
Cluster input	Independent	Tables	2	(56,0)	0,000	(0,28)	0,000	(56,0)	0,000	(0,28)	0,000									0,000	0,000	0,000
		Columns	5	(50,0,2,2,2)	-0,458	(0,28,0,0,0)	0,000	(50,0)	0,000	(0,28)	0,000	(2,0)	0,000	(2,0)	0,000	(2,0)	0,000			-0,458	0,000	-0,458
		Operations	2	(56,0)	0,000	(0,28)	0,000	(56,0)	0,000	(0,28)	0,000									0,000	0,000	0,000
		Model Property Values	1	(56)	0,000	(28)	0,000	(56,28)	-0,637											0,000	-0,637	-0,637
	Pair	Tables and Columns	5	(50,0,2,2,2)	-0,458	(0,28,0,0,0)	0,000	(50,0)	0,000	(0,28)	0,000	(2,0)	0,000	(2,0)	0,000	(2,0)	0,000			-0,458	0,000	-0,458
		Tables and Operations	2	(56,0)	0,000	(0,28)	0,000	(56,0)	0,000	(0,28)	0									0,000	0,000	0,000
		Tables and Model Property Values	2	(56,0)	0,000	(0,28)	0,000	(56,0)	0,000	(0,28)	0,000									0,000	0,000	0,000
		Columns and Operations	2	(56,0)	0,000	(0,28)	0,000	(56,0)	0,000	(0,28)	0,000									0,000	0,000	0,000
		Columns and Model Property Values	4	(14,28,14,0)	-1,040	(0,0,0,28)	0,000	(14,0)	0,000	(28,0)	0,000	(14,0)	0,000	(0,28)	0,000					-1,040	0,000	-1,040
		Operations and Model Property Values	3	(28,0,28)	-0,693	(0,28,0)	0,000	(28,0)	0,000	(0,28)	0,000	(28,0)	0,000							-0,693	0,000	-0,693
	Triple	Tables, Columns and Operations	2	(0,56)	0,000	(28,0)	0,000	(0,28)	0,000	(56,0)	0,000									0,000	0,000	0,000
		Tables, Columns and Model Property Values	2	(56,0)	0,000	(0,28)	0,000	(56,0)	0,000	(0,28)	0,000									0,000	0,000	0,000
		Tables, Operations and Model Property Values	6	(1,16,3,23,0,13)	-1,291	(12,0,0,0,16,0)	-0,683	(1,12)	-0,271	(16,0)	0	(3,0)	0	(23,0)	0	(0,16)	0	(13,0)	0	-1,974	-0,271	-2,245
		Columns, Operations and Model Property Values	6	(4,20,0,0,24,8)	-1,197	(0,0,14,14,0,0)	-0,693	(4,0)	0,000	(20,0)	0,000	(0,14)	0,000	(0,14)	0,000	(24,0)	0,000	(8,0)	0,000	-1,890	0,000	-1,890
	All	Tables, Columns, Operations and Model Property Values	3	(28,0,28)	-0,693	(0,28,0)	0,000	(28,0)	0,000	(0,28)	0,000	(28,0)	0,000							-0,693	0,000	-0,693

TABLE XII
ENTROPY DETAILS FOR TEST 4: FOR EACH TYPE OF INPUT THE DEVIATION ENTROPY IS EVALUATED PER TYPE OF DEVIATION AND THE CLUSTER ENTROPY IS EVALUATED PER CLUSTER

		Test 4	Number of clusters	Fault 50030		Change 50031		Change 48580		Cluster 0		Cluster 1		Cluster 2		Cluster 3		Cluster 4		Total Deviation Entropy	Total Cluster Entropy	Total Entropy
				Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy	Distribution	Entropy			
Cluster input	Independent	Tables	3	(0,0,3)	0,000	(0,10,0)	0,000	(30,0,0)	0,000	(0,0,30)	0,000	(0,10,0)	0,000	(3,0,0)	0,000					0,000	0,000	0,000
		Columns	5	(0,0,3,0,0)	0,000	(0,0,10,0,0)	0,000	(5,9,0,12,4)	-1,295	(0,0,5)	0,000	(0,0,9)	0,000	(3,10,0)	-0,540	(0,0,12)	0,000	(0,0,4)	0,000	-1,295	-0,540	-1,835
		Operations	5	(0,0,0,0,3)	0,000	(0,10,0,0,0)	0,000	(25,0,3,2,0)	-0,563	(0,0,25)	0,000	(0,10,0)	0,000	(0,0,3)	0,000	(0,0,2)	0,000	(3,0,0)	0,000	-0,563	0,000	-0,563
		Model Property Values	3	(1,0,2)	-0,637	(1,0,9)	-0,325	(19,11,0)	-0,657	(1,1,19)	-0,381	(0,0,11)	0,000	(2,9,0)	-0,474					-1,619	-0,855	-2,473
	Pair	Tables and Columns	5	(0,0,0,0,3)	0,000	(0,0,10,0,0)	0,000	(12,9,0,9,0)	-1,089	(0,0,12)	0,000	(0,0,9)	0	(0,10,0)	0,000	(0,0,9)	0	(3,0,0)	0	-1,089	0,000	-1,089
		Tables and Operations	5	(0,0,0,0,3)	0,000	(0,10,0,0,0)	0,000	(12,0,3,15,0)	-0,943	(0,0,12)	0,000	(0,10,0)	0	(0,0,3)	0	(0,0,15)	0	(3,0,0)	0	-0,943	0,000	-0,943
		Tables and Model Property Values	2	(3,0)	0,000	(10,0)	0,000	(0,30)	0,000	(3,10,0)	-0,540	(0,0,30)	0,000							0,000	-0,540	-0,540
		Columns and Operations	5	(0,0,0,0,3)	0,000	(0,0,10,0,0)	0,000	(12,9,0,9,0)	-1,089	(0,0,12)	0,000	(0,0,9)	0,000	(0,10,0)	0,000	(0,0,9)	0,000	(3,0,0)	0,000	-1,089	0,000	-1,089
		Columns and Model Property Values	4	(0,0,3,0)	0,000	(0,0,10,0)	0,000	(12,11,0,7)	-1,074	(0,0,12)	0,000	(0,0,11)	0,000	(3,10,0)	-0,540	(0,0,7)	0,000			-1,074	-0,540	-1,614
		Operations and Model Property Values	2	(0,3)	0,000	(0,10)	0,000	(30,0)	0,000	(0,0,30)	0,000	(3,10,0)	-0,540							0,000	-0,540	-0,540
	Triple	Tables, Columns and Operations	5	(0,0,0,0,3)	0,000	(0,0,10,0,0)	0,000	(12,9,0,9,0)	-1,089	(0,0,12)	0,000	(0,0,9)	0,000	(0,10,0)	0,000	(0,0,9)	0,000	(3,0,0)	0,000	-1,089	0,000	-1,089
		Tables, Columns and Model Property Values	3	(3,0,0)	0,000	(10,0,0)	0,000	(0,18,12)	-0,673	(3,10,0)	-0,540	(0,0,18)	0,000	(0,0,12)	0,000					-0,673	-0,540	-1,213
		Tables, Operations and Model Property Values	4	(0,0,3,0)	0,000	(0,0,10,0)	0,000	(12,4,0,14)	-0,991	(0,0,12)	0,000	(0,0,4)	0	(3,10,0)	-0,54	(0,0,14)	0			-0,991	-0,540	-1,531
		Columns, Operations and Model Property Values	2	(3,0)	0,000	(10,0)	0,000	(0,30)	0,000	(3,10,0)	-0,540	(0,0,30)	0,000							0,000	-0,540	-0,540
	All	Tables, Columns, Operations and Model Property Values	4	(0,0,0,3)	0,000	(0,0,0,10)	0,000	(9,12,9,0)	-1,089	(0,0,9)	0,000	(0,0,12)	0,000	(0,0,9)	0,000	(3,10,0)	-0,540			-1,089	-0,540	-1,629

Acknowledgment

Lionel Briand was supported by the National Research Fund, Luxembourg (FNR/P10/03), while Erik Rogstad was supported by the ATOS project, a joint project of The Norwegian Tax Department and Simula Research Laboratory. The authors would also like to thank Cu Nguyen and Claudia Thür for their feedback on the paper.

References

[1] M. Harrold and A. Orso, "Retesting software during development and maintenance," in *Frontiers of Software Maintenance, 2008 (FoSM 2008)*, 2008, pp. 99–108.

[2] S. Yoo and M. Harman, "Regression testing minimisation, selection and prioritisation: A survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, March 2012.

[3] E. Rogstad, L. Briand, E. Arisholm, R. Dalberg, and M. Rynning, "Industrial experiences with automated regression testing of a legacy database application," in *27th IEEE International Conference on Software Maintenance (ICSM)*, September 2011, pp. 362–371.

[4] E. Lehmann and J. Wegener, "Test case design by means of the CTE-XL," in *Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000)*, December 2000.

[5] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, Oct 2009, pp. 401–404.

[6] E. Rogstad and L. Briand, "Test case selection for black-box regression testing of database applications," *Information and Software Technology (IST)*, vol. 31, no. 6, pp. 676–686, June 2013.

[7] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco: Morgan Kaufmann, 2005.

[8] G. McLachlan and T. Krishnan, *The EM algorithm and extensions*, 2nd ed. Hoboken, NJ: Wiley, 2008.

[9] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Engg.*, vol. 14, no. 2, pp. 131–164, April 2009. [Online]. Available: http://dx.doi.org/10.1007/s10664-008-9102-8

[10] R. M. Gray, *Entropy and Information Theory*, 2nd ed. New York: Springer, 2011.

[11] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," in *33rd International Conference on Software Engineering (ICSE)*, May 2011, pp. 1–10.

[12] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[13] R. K. Yin, *Case study research: Design and methods*, 3rd ed. London: Sage, 2003.

[14] C. Graham, Y. Dharmasthira, C. Eschinger, B. F. Granetto, J. M. Correia, L. F. Wurster, F. D. Silva, T. Eid, R. Contu, F. Biscotti, C. Pang, D. M. Coyle, D. Sommer, M. Cheung, H. H. Swinehart, B. Sood, A. Raina, K. Motoyoshi, Y. Nagashima, J. Zhang, M. Warrilow, S. Deshpande, J.-S. Yim, V. Mladjov, and J. Mazzucca. (2014, March) Market share: All software markets, worldwide, 2013. [Online]. Available: https://www.gartner.com/doc/2695617

[15] M. Borg and P. Runeson, "Ir in software traceability: From a bird's eye view," in *Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on*, Oct 2013, pp. 243–246.

[16] W. Dickinson, D. Leon, and A. Podgurski, "Finding failures by cluster analysis of execution profiles," in *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*, 2001, pp. 339–348.

[17] A. Podgurski and C. Yang, "Partition testing, stratified sampling, and cluster analysis," *SIGSOFT Softw. Eng. Notes*, vol. 18, no. 5, pp. 169–181, Dec. 1993. [Online]. Available: http://doi.acm.org/10.1145/167049.167076

[18] A. Podgurski, W. Masri, Y. McCleese, F. G. Wolff, and C. Yang, "Estimation of software reliability by stratified sampling," *ACM Trans. Softw. Eng. Methodol.*, vol. 8, no. 3, pp. 263–283, Jul. 1999. [Online]. Available: http://doi.acm.org/10.1145/310663.310667

[19] S. Parsa, A. Khalilian, and Y. Fazlalizadeh, "A new algorithm to test suite reduction based on cluster analysis," in *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, 2009, pp. 189–193.

[20] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Trans. Softw. Eng. Methodol.*, vol. 2, no. 3, pp. 270–285, Jul. 1993. [Online]. Available: http://doi.acm.org/10.1145/152388.152391

[21] V. Vangala, J. Czerwonka, and P. Talluri, "Test case comparison and clustering using program profiles and static execution," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 293–294. [Online]. Available: http://doi.acm.org/10.1145/1595696.1595748

[22] C. Zhang, Z. Chen, Z. Zhao, S. Yan, J. Zhang, and B. Xu, "An improved regression test selection technique by clustering execution profiles," in *Quality Software (QSIC), 2010 10th International Conference on*, 2010, pp. 171–179.

[23] 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, Apr. 1997. [Online]. Available: http://doi.acm.org/10.1145/248233.248262

[24] S. Chen, Z. Chen, Z. Zhao, B. Xu, and Y. Feng, "Using semi-supervised clustering to improve regression test selection techniques," in *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, 2011, pp. 1–10.

[25] P. G. Sapna and H. Mohanty, "Clustering test cases to achieve effective test selection," in *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*, ser. A2CWiC '10. New York, NY, USA: ACM, 2010, pp. 15:1–15:8. [Online]. Available: http://doi.acm.org/10.1145/1858378.1858393

[26] Y. Liu, K. Wang, W. Wei, B. Zhang, and H. Zhong, "User-session-based test cases optimization method based on agglutinate hierarchy clustering," in *Proceedings of the 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, ser. ITHINGSCPSCOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 413–418. [Online]. Available: http://dx.doi.org/10.1109/iThings/CPSCom.2011.135

[27] Y. Wang, Z. Chen, Y. Feng, B. Luo, and Y. Yang, "Using weighted attributes to improve cluster test selection," in *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on*, 2012, pp. 138–146.

[28] S. Yan, Z. Chen, Z. Zhao, C. Zhang, and Y. Zhou, "A dynamic test cluster sampling strategy by leveraging execution spectra information," in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, 2010, pp. 147–154.

[29] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *Proceedings of the eighteenth international symposium on Software testing and analysis*, ser. ISSTA '09. New York, NY, USA: ACM, 2009, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/1572272.1572296

[30] C. Simons and E. Paraiso, "Regression test cases prioritization using failure pursuit sampling," in *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, 2010, pp. 923–928.

[31] R. Carlson, H. Do, and A. Denton, "A clustering approach to improving test case prioritization: An industrial case study," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, 2011, pp. 382–391.

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

[33] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated support for classifying software failure reports," in *Software Engineering, 2003. Proceedings. 25th International Conference on*, 2003, pp. 465–475.

[34] C.-H. Hsueh, Y.-P. Cheng, and W.-C. Pan, "Intrusive test automation with failed test case clustering," in *Software Engineering Conference (APSEC), 2011 18th Asia Pacific*, 2011, pp. 89–96.

[35] N. DiGiuseppe and J. A. Jones, "Concept-based failure clustering," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 29:1–29:4. [Online]. Available: http://doi.acm.org/10.1145/2393596.2393629

[36] L. Kaufman and P. J. Rousseeuw, *Finding groups in data - An introduction to cluster analysis*. United States: John Wiley and Sons Inc., 2005.

[37] T. Calinski and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics*, vol. 3, no. 1, pp. 1–27, 1974.

[38] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443 – 453, 1970. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022283670900574

Erik Rogstad is a PhD student at Simula Research Laboratory in Norway. He works within the Software Engineering department, focusing on Software Testing.

Erik did his Master in Computer Science at the Norwegian University of Science and Technology in 2006, and worked as an IT consultant for Capgemini prior to joining Simula for a Ph.D. position. After the submission of the Ph.D. thesis, Erik will join Testify AS, to work as a consultant dedicated towards software testing.

Mail address:
Erik Rogstad
Simula Research Laboratory
P.O. Box 134
1325 Lysaker
NORWAY

Lionel C. Briand is professor and FNR PEARL chair in software verification and validation at the SnT centre for Security, Reliability, and Trust, University of Luxembourg. He also acts as vice-director of the centre. Lionel started his career as a software engineer in France (CS Communications & Systems), and has conducted applied research in collaboration with industry for more than 20 years.

Until moving to Luxembourg in January 2012, he was heading the Certus center for software verification and validation at Simula Research Laboratory, where he was leading applied research projects in collaboration with industrial partners. Before that, he was on the faculty of the department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he was full professor, and held the Canada Research Chair (Tier I) in Software Quality Engineering. He has also been the software quality engineering department head at the Fraunhofer Institute for Experimental Software Engineering, Germany; and worked as a research scientist for the Software Engineering Laboratory, a consortium of the NASA Goddard Space Flight Center, CSC, and the University of Maryland, USA.

Lionel was elevated to the grade of IEEE Fellow for his work on the testing of object-oriented systems. He was recently granted the IEEE Computer Society Harlan Mills award, and the IEEE Reliability Society engineer-of-the-year award for his work on model-based verification and testing. His research interests include software testing and verification, model-driven software development, search-based software engineering, and empirical software engineering.

Lionel has been on the program, steering, or organization committees of many international, IEEE, and ACM conferences. He is the coeditor-in-chief of Empirical Software Engineering (Springer); and is a member of the editorial boards of Systems and Software Modeling (Springer); and Software Testing, Verification, and Reliability (Wiley).

Mail address:
Lionel Briand
University of Luxembourg
SnT Centre
4, rue Alphonse Weicker
L-2721 Luxembourg
LUXEMBOURG