Sanitizing And Minimizing Databases For Software Application Test Outsourcing

Boyang Li
College of William and Mary
Williamsburg, VA 23185
boyang@cs.wm.edu

Mark Grechanik
University of Illinois at Chicago
Chicago, IL 60607
drmark@uic.edu

Denys Poshyvanyk
College of William and Mary
Williamsburg, VA 23185
denys@cs.wm.edu

***Abstract*—Testing software applications that use nontrivial databases is increasingly outsourced to test centers in order to achieve lower cost and higher quality. Not only do different data privacy laws prevent organizations from sharing this data with test centers because databases contain sensitive information, but also this situation is aggravated by *big data* – it is time consuming and difficult to anonymize, distribute, and test with large databases. Deleting data randomly often leads to significantly worsened test coverages and fewer uncovered faults, thereby reducing the quality of software applications.**

We propose a novel approach for *Protecting and mInimizing databases for Software TestIng taSks (PISTIS)* that both sanitizes and minimizes a database that comes along with an application. PISTIS uses a weight-based data clustering algorithm that partitions data in the database using information obtained using program analysis that describes how this data is used by the application. For each cluster, a centroid object is computed that represents different persons or entities in the cluster, and we use associative rule mining to compute and use constraints to ensure that the centroid objects are representative of the general population of the data in the cluster. Doing so also sanitizes information, since these centroid objects replace the original data to make it difficult for attackers to infer sensitive information. Thus, we reduce a large database to a few centroid objects and we show in our experiments with two applications that test coverage stays within a close range to its original level.

I. Introduction

Database-centric applications (DCAs) are common in enterprise computing, and they use nontrivial databases [26]. Testing DCAs is increasingly outsourced to test centers in order to achieve lower cost and higher quality [5], [15]. Unfortunately, not only do different data privacy laws prevent organizations from sharing this data with test centers because databases contain sensitive information [41], [45], but it is also very time consuming and difficult to anonymize, distribute, and test applications with large databases. Complying with these privacy laws by removing and sanitizing data often leads to significantly worsened test coverages and fewer uncovered faults, thereby reducing the quality of software applications [21]. For instance, if values of the attribute `Nationality` are replaced with the generic value “`Human`,” DCAs may execute some paths that result in exceptions or miss certain paths [21]. As a result, test centers report worse test coverage (e.g., statement coverage) and fewer uncovered faults, thereby reducing the quality of DCAs and obliterating benefits of test outsourcing in distributed software development [33].

This situation is aggravated by *big data*—collections of large-sized data sets that contain patterns that may be useful for some tasks [6], [7], [23], [28]. To perform these tasks using big data, DCAs use databases whose sizes are measured in hundreds of terabytes on the low end. Our interviews with contractors who use industry-strength tools like IBM Optim[1] reveal that sanitizing large databases often takes many weeks and requires significant resources. In addition, maintaining and resetting states of large databases when testing DCAs is difficult [18], [22]. Ideally, the size of a database should be reduced to alleviate testing without sacrificing its quality.

Sanitizing and Minimizing (S&M) databases are loosely connected tasks – in some cases, removing data from a database that describe persons or entities may hide sensitive information about them. However, in general, minimizing databases does not hide sensitive information and sanitizing data does not result in smaller databases. A fundamental problem in test outsourcing is how to allow a DCA owner to release a smaller subset of its private data with guarantees that the entities in this data (e.g., people, organizations) are protected at a certain level while retaining testing efficacy. Ideally, sanitized (or anonymized) data should induce execution paths that are similar to the ones that are induced by the original data. In other words, when databases are S&Med, information about how DCAs use this data should be taken into consideration. In practice, this consideration rarely happens; our previous work [21] as well as follow-up research [8], [36] showed that many data anonymization algorithms seriously degrade test coverages of DCAs.

To address this issue, we offer a novel approach for *Protecting and mInimizing databases for Software TestIng taSks (PISTIS)* that both sanitizes and minimizes a database that comes along with an application. PISTIS uses a weight-based data clustering algorithm that partitions data in the database using information from program analysis in weights that indicate how this data is used by the application. For each cluster, a centroid object is computed that represents different persons or entities in the cluster, and we use associative rule mining to compute and use constraints to ensure that the centroid objects are representative of the general population of data in the cluster. At the same time, we sanitize information

[1]http://www-01.ibm.com/software/data/optim

in these centroid objects to make it difficult for attackers to infer sensitive information. Our paper makes the following contributions.

- We created a novel approach that achieves both sanitization and minimization of data using a novel combination of data mining approaches, program analysis, and privacy constraints. PISTIS is platform and application and database-neutral; it is widely applicable to different DCAs that use different databases, especially in the context of big data. Our work is unique; to the best of our knowledge, there exists no prior approach that synergetically addresses all components of the S&M problem that we pose in this paper.
- We evaluated our approach on two open-source Java applications and we show in our experiments that a reduction in statement coverage of no more than 25%, while minimizing the size of the database by more than an order of magnitude.
- All data files and subject applications are available for reproducibility from the project website http://www.cs.uic.edu/~drmark/PISTIS.htm.

II. An Illustrative Example

In this section, we describe how PISTIS works using an illustrative example.

A. The DCA and Its Database

The original database is shown as a table on the left side of Figure 2 and the resulting S&M database is shown to the right side of the arrow after applying PISTIS to the original database to S&M it. A DCA that uses this database is shown as a fragment of Java-like pseudo-code in Figure 1. Line numbers to the right should be thought of as labels, as much code is omitted for space reasons. This example has `if-else` statements that control six branches, where branch labels are shown in comments along with the numbers of statements that these branches control. These numbers are given purely for illustrative purposes. We omit database connectivity code that retrieves data from the attributes `Age`, `Gender`, and `Treatment` of the database table and puts this data into the corresponding variables `Age`, `Gender`, and `Treatment`. The last column in Figure 2 shows what branches are covered in the DCA with the data that belong to specific rows.

In our illustrative example, when a branch condition of a control-flow statement is evaluated to `true`, some code in the branch is executed in the scope of this statement. The statements that are contained in the executed code are said to be controlled by or contained in the corresponding branch of this control-flow statement. In program analysis, these statements are said to be *control dependent* on the control-flow statement [32]. Conditions of the control-dependent branches use variables that contain data from the corresponding attributes of the database, and it means that these values control branch coverage. For example, if the values of the attribute `Gender` are sanitized by replacing the values “`Male`” and “`Female`” with “`Human`”, the resulting branch code coverage

```
if( Age >= 18 && Gender == "Male" ) {                   1
  .. // branch B1: 100 statements                       2
  if( Treatment == "Chemotherapy" ) {                   3
  .. // branch B3: 100 statements                       4
  } else if( Treatment == "Vasectomy" ) {               5
        .. // branch B4: 100 statements                 6
  }                                                     7
} else if( Age < 70 && Gender == "Female" ) {           8
  .. // branch B2: 100 statements                       9
  if( Treatment == "Chemotherapy" ) {                   10
  .. // branch B5: 100 statements                       11
  } else if( Treatment == "Hysterectomy" ) {            12
        .. // branch B6: 100 statements                 13
} }                                                     14
```

Fig. 1: An illustrative example of Java pseudocode that uses the data shown in Figure 2. For the sake of simplicity, variables are given names that match attributes.

will be zero when testing the code in Figure 1 using this sanitized data. Reducing the size of the database by randomly removing six records out of nine will result in the reduced branch coverage by up to 66.7% (i.e., removing all four records for males and the two records for females who underwent chemotherapy).

B. Normalizing the Data

PISTIS involves three main steps. The first step involves transformation of the database, so that distance can be computed between different types of data. This step is shown in Figure 3, where each column represents distinct values for database attributes from the original database. Essentially, clustering data is based on computing distances between different types of data. For numerical attributes, such as `Age`, computing the distance is trivial by obtaining the absolute value of the difference between two age values. In addition, we normalize this value by dividing the maximum value of `Age` in the database. The original and normalized values of Age are shown in the columns $\texttt{Age}_o$ and $\texttt{Age}_n$ respectively.

For categorical attributes, the situation is different, since it is not easy in general to define the order on the values of these attributes or enumerate them in a specific order. In this paper, we follow a general approach to assume that every distinct value for a given attribute is equally different from one another [9]. We represent each distinct value either as one if it is present for a given row, or zero if it is not present. This is the essence of the step one in constructing the intermediate representation that is shown in Figure 3.

C. Clustering the Database

The second step is to cluster data by grouping them in a way that similar entities (i.e., objects or persons) are described by the data that are located in the same cluster. We define the meaning of the similarity between entities by using Mizzaro's well-established conceptual framework for relevance [30], [31]. In Mizzaro's framework, similar entities are relevant to one another if they share some common properties. Once these properties are known, entities can be clustered by how they

Rec	Age	Zip	Gender	Treatment	Branch
1	42	53000	Male	Vasectomy	B1,B4
2	47	53000	Female	Hysterectomy	B2,B6
3	51	32000	Male	Chemotherapy	B1,B3
4	55	32000	Male	Chemotherapy	B1,B3
5	62	53000	Female	Chemotherapy	B2,B5
6	67	35000	Female	Hysterectomy	B2,B6
7	30	53000	Male	Vasectomy	B1,B4
8	31	35000	Female	Chemotherapy	B2,B5
9	35	53000	Female	Hysterectomy	B2,B6

$\Rightarrow$

Rec	Age	Zip	Gender	Treatment	Branch
1	47	53000	Male	Vasectomy	B1,B4
2	61	53000	Female	Chemotherapy	B2,B5
3	32	52000	Female	Hysterectomy	B2,B6

Fig. 2: Transformation of the original table on the left into a sanitized table on the right that contains centroid data. The last column, `Branch`, designates covered branches in the example shown in Figure 1 for each row.

Rec	Age_o	Age_n	53000	32000	35000	Female	Male	Hysterectomy	Vasectomy	Chemotherapy
1	42	0.63	1	0	0	0	1	0	1	0
2	47	0.7	1	0	0	1	0	1	0	0
3	51	0.76	0	1	0	0	1	0	0	1
4	55	0.82	0	1	0	0	1	0	0	1
5	62	0.93	1	0	0	1	0	0	0	1
6	67	1.0	0	0	1	1	0	1	0	0
7	30	0.45	1	0	0	0	1	0	1	0
8	31	0.46	0	0	1	1	0	0	0	1
9	35	0.52	1	0	0	1	0	1	0	0

Fig. 3: Step 1 of the transformation: translate the original data table into a normalized table. Step 2: cluster the normalized table using weighted k-means clustering where $k = 3$. Clusters are shown with double horizontal dividers that separate rows.

Rec	Age_o	Age_n	53000	32000	35000	Female	Male	Hysterectomy	Vasectomy	Chemotherapy
C_1	47	0.7	0.67	0.33	0	0.33	0.67	0.33	0.33	0.33
C_2	61	0.92	0.33	0.33	0.33	0.67	0.33	0	0.33	0.67
C_3	32	0.48	0.67	0	0.33	0.67	0.33	0.33	0.33	0.33

Fig. 4: Step 3 of the transformation: compute centroids for clusters.

Original Centroid	Record 1	Record 2	Record 3	Record 4	Record 5	Record 6	Record 7	Record 8	Record 9
C_1	0.41	0.26	0.25	0.25	0.24	0	0.39	0.01	0.25
C_2	0.03	0.03	0.18	0.18	0.35	0.04	0	0.3	0.01
C_3	0.25	0.4	0.01	0	0.36	0.2	0.26	0.26	0.41

Fig. 5: A similarity matrix where cells contain the values that show the fraction of attributes whose values that are the same between original and centroid records. Value zero means that the record does not match any centroids and value one means that there is a total match, where the original and centroid records are the same.

are similar by comparing these properties, i.e., the values of their attributes. Subsequently all entities in each cluster will be more similar to one another when compared to entities that belong to other clusters. This is the essence of the cluster hypothesis that specifies that data that cluster together tend to share the same or closely related values of their attributes [46]. Our intuition is that since all entities in a cluster are similar to one another, they may have the same effect on the DCA that uses these entities and thus only one representative of the cluster is needed to test this DCA.

However, in general, attributes do not have equal weights. Clearly, the values of the attributes `Age`, `Gender`, and `Treatment` control branches of the code that is shown in Figure 1, while the attribute `Zip` has no effect on this DCA, since the latter does not use it at all. Therefore, when clustering this database, the similarity between the values of `Age` for two different entities may play a bigger role when compared with the same values for the attribute `Zip`. Consequently, we compute weights for different attributes by approximating how their values affect the execution of the DCA. We discuss how we compute weights in Section III.

Using this approach, we cluster the database and the results are shown in Figure 3, where clusters are designated using double-separator lines between rows. Of course, clustering is an approximation of assigning different entities to different groups. A cursory study reveals that entities designated by rows are more similar to one another in each cluster – the ages and genders for each person in the cluster are closer to one another when compared to persons from the other clusters. Moreover, we notice that most data that belong to the same cluster leads to the same branch coverage of the DCA.

D. Computing Centroids

The third step is to compute centroid entities (i.e., *centroids*), which are records that represent data in clusters. We do so by computing the average of values for each column for each cluster in Figure 3. Once it is done, we map the resulting values back to specific distinct values and break ties by randomly picking one. However, doing so may result in semantically incorrect entities. Consider a case when a centroid is created that describes a male who underwent hysterectomy, a procedure that can only be done for female patients. Semantically meaningless data poses a significant problem for software testing.

Original data elements are often connected via intricate semantic relationships that are not always explicitly defined in database schemas. Testing with synthetic data often does not yield the same results compared to testing with real data. Consider one aspect of testing – comparing results with oracles, automatic generation of which is a fundamentally difficult problem [17] [35] [39] [40]. In many cases, domain experts review results of testing manually, since generated data often leaves the expected results unspecified [25, pages 114, 116]. When important relationships are missed between data elements, running applications with test cases that use this synthetic data produces results that make little sense to these domain experts. For example, it may not make any sense to develop oracles for medical insurance software if the generated input test data describes a male who underwent hysterectomy, for whom computing insurance premium (the oracle) is not applicable. In reality, there are multiple relationships among data elements, many of which are far from obvious. Thus, it is important to release semantically correct data to testers.

E. Using Association Rules For Semantic Correctness

To reduce the chance of creating semantically incorrect centroids, we obtain association rules that in general describe relations between different elements as implication expressions $A_1 \wedge A_2 \wedge \ldots \wedge A_i \Rightarrow C_1 \vee C_2 \vee \ldots \vee C_j$, where A and C are disjoint itemsets, that is $A \cap C = \emptyset$ [42, pages 327–332]. There are different algorithms for extracting association rules from databases, and the quality metrics are support and confidence. Support measures the applicability of an association rule to a dataset, and confidence estimates the frequency with which items in C also appear in query results that contain A. Support, s, and confidence, c are calculated using the following formulae: $s(A \Rightarrow C) = \frac{\sigma(A \cup C)}{N}$ and $c(A \Rightarrow C) = \frac{\sigma(A \cup C)}{\sigma(A)}$, where σ is the number of support records for the parameter and N is the total number of records. In our case, we extract two associative rules: `Hysterectomy` $\Rightarrow$ `Female` and `Vasectomy` $\Rightarrow$ `Male`. Using these rules, we take computed centroids that are shown in Figure 4 and translate them into original distinct values that are shown on the right side in Figure 2. Doing so, we partially solve the sanitization and minimization problem: we sanitized the original data and we reduced the size of the database by two thirds. Also, all branches but B3 are covered with the centroids.

F. Data Privacy

Finally, a question remains how easy it is for an attacker to guess the original data given the sanitized data. A privacy metric measures how identifiable records in the sanitized table are with respect to the original table [19, page 43]. Our idea is to quantify identifiability of records using a similarity matrix that shows the similarity between sanitized (centroids) and original records. For each record R_o in the original table, the similarity of R_o to a record C_i in the sanitized table is measured by the fraction of attributes whose values are the same between R_o and C_i. Table 5 shows an illustrative example for computing the similarity matrix $\|D\|$ that has dimensions $r \times c$, where r is the number of records in original table and c is the number of records in the centroid table. Rows correspond to centroid records and columns correspond to the original records in $\|D\|$.

We use the similarity matrix $\|D\|$ to compute how difficult it is for attackers to guess original records given sanitized records. Consider an extreme case where all entries are zero. In this case, each record is dissimilar to every centroid, that is, all records are fully protected. On the other hand, if all diagonal entries are one and other entries are zero, then each record is unique and easily identifiable by attackers. In our case, it is very difficult for attackers to guess original data since all centroid records are dissimilar to all original records. Entries marked one should be dealt separately; they should either be deleted from the database with possible reduction of the test coverage or left in the database with a calculated risk for disclosure of this potentially sensitive information. One way or the other, stakeholders can make a calculated decision about the balance between test coverage, privacy, and data minimization.

III. Our Solution

In this section, we present core ideas behind PISTIS and we describe its architecture and the workflow.

A. Core Ideas

At the core of our work are three major ideas. The first one is our approach for sanitizing data that enables organizations to keep derivatives of the original values in sanitized data (i.e., centroids). As a result, test coverage is not affected so negatively as it happens when data suppression and generalization techniques are used. In addition, semantic integrity of the sanitized data is largely preserved, since we use associative rule mining to obtain constraints from the original data. Coincidentally, reducing the original database to centroids results in minimizing the database, thereby partially solving the S&M problem using a single approach.

The second idea is our guessing anonymity privacy metric that allows stakeholders to quantify the level of privacy achieved in a sanitized database. In particular, the metric provides measurement of difficulty for an attacker to relate a sanitized record to the original record. We apply the idea that we previously developed for PRIEST in this new S&M context [43].

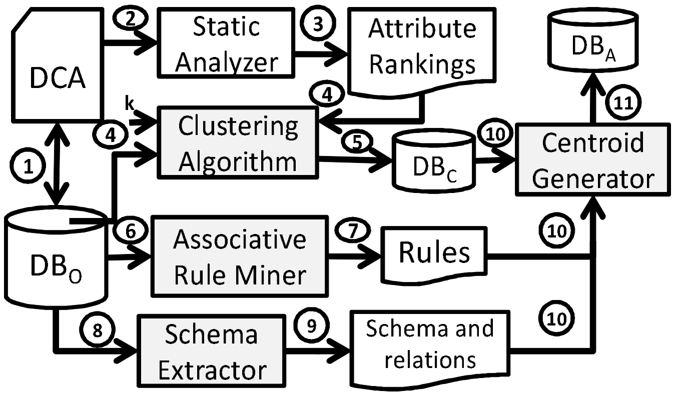

Fig. 6: PISTIS architecture and workflow. Solid arrows depict the flow of command and data between components, numbers indicate the sequence of operations in the workflow.

The third idea is an idea to improve the precision of computing centroids in a way that the sanitized databases maintain testing utility. We statically determine how different database values affect the behavior of a DCA. This idea unifies DCAs and their databases in a novel way – database attributes are tied to the source code of the DCAs, and depending on how the DCAs use values of these attributes, clustering finds similar groups of data from which centroids are computed without sacrificing much of test coverage.

B. PISTIS Architecture and Process

The first step of the PISTIS process involves programmers who link program variables to database attributes using annotations, so that these annotations can be traced statically using control- and data-flow analyses. Tracing these attribute annotations is required to determine how the values of these attributes are used in conditional expressions to make branching decisions, thereby influencing the execution flow of the DCAs. Quantifying the effect of replacing values of database attributes on reachability of program statements enables us to apply weighted *k*-mean clustering to group database records.

Figure 6 shows the architecture of PISTIS. The inputs to PISTIS are the application's source code and the original database DB_O that this DCA uses `(1)`. PISTIS performs control- and data-flow analyses `(2)` using the Soot toolkit[2] to establish how the DCA uses values of different database attributes. Values of some attributes are used in expressions to compute other values, which in turn are used in other expressions and statements. In some cases, these values are used in conditional statements, and they affect control flows of DCAs using control-flow dependencies. Ideally, attributes whose values affect many other expressions and statements in DCAs (in terms of branch coverage) should be assigned the highest weights for clustering. The output `(3)` of this procedure is a list of attribute rankings that show how many statements are approximately encapsulated by branches whose conditions contain program variables that receive their values from database attributes.

[2]http://www.sable.mcgill.ca/soot

The next step is to run the Clustering Algorithm, which takes as its input `(4)` the number of clusters (it is a user-defined parameter), the original database DB_O, and the Attribute Rankings. Once clustering is done, `(5)` the clustered original database DB_C is outputted. Then, `(6)` the Associative Rule Miner computes `(7)` rules that describe semantic constraints that are obtained from DB_O.

At this point, PISTIS `(8)` extracts `(9)` the database schema and its constaints from DB_O using JDBC metadata services. The schema and constaints are important in conjunction with rules to ensure that centroids are compliant with the schema and the constraints. For example, if an attribute is indexed and its values are unique, putting duplicate values may lead to exceptions within the DCA. In addition, foreign keys and dependencies ensure the match among values of certain attributes in the database. Violating the schema and its constrains reduces the efficacy of testing.

Finally, `(10)` the Rules and the Schema and Relations along with the clustered database DB_C is inputted into the Centroid Generator that `(11)` computes the resulting anonymized database DB_A that contains centroids.

C. Ranking Attributes

To understand which attributes affect DCAs the most, we rank these attributes by counting the numbers of statements that their values affect. To find the preceding information, our approach uses static taint analysis to track the annotated variables corresponding to each attribute [43]. In particular, for each attribute, our approach uses control- and data-flow taint propagation [13] to find out branch conditions that are tainted by an annotated variable corresponding to the attribute.

Specifically, we construct and traverse a *control-flow graph (CFG)* of the DCA. When traversing the CFG we count the number of statements and branches that are control-dependent on branch conditions that are linked to database attributes. We perform virtual-call resolution using static class hierarchy analysis, and we take a conservative approach by counting the biggest number of statements of a method that can potentially be invoked. We also count all the statements in all the target methods that can potentially be invoked, but only when the call site is the only entry point of that method. Currently, we only take into consideration that values of attributes are used in variables that control branches.

IV. Experimental Evaluation

In this section, we describe the results of the experimental evaluation of PISTIS on two open-source Java programs.

A. Research Questions

In this paper, we make a claim that using PISTIS solves the S&M problem, i.e., it enables stakeholders to both reduce the size of the database and sanitize data while maintaining testing utility. We seek to answer the following research questions.

RQ1 How much test coverage does PISTIS help achieve at given levels of disclosure risk?

RQ2 How effective is PISTIS in reducing the size of the database for maintaining test coverage?

RQ3 How effective is PISTIS in reducing the size of the database for low disclosure rates for certain levels of test coverage?

With these RQs we decompose our experimental results to evaluate the effectiveness of PISTIS for different components of the S&M problem. With RQ1, we address our claim that we designed and implemented a technique that combines program analysis for determining how values of database attributes affect test coverage of DCAs with data mining techniques for creating centroids that hide original data thus reducing the disclosure risk. Our goal is to show that with PISTIS, the disclosure risk metric is linked directly to test coverage and vice versa; in other words, guaranteeing a certain level of test coverage should allow stakeholders to estimate bounds of the privacy level.

With RQ2, we address our claim that it is possible to reduce the size of the database while at the same time maintaining testing efficacy. Since competitive approaches for minimization of data use data compression or deletion techniques, which are independent of sanitization, PISTIS is in itself a contribution, since it enables stakeholders to make trade-off decisions about data minimization and testing utility. In that, our goal is to show that minimizing data using PISTIS enables stakeholders to maintain higher levels of test coverage, which will not be possible with data compressing and deletion techniques, since they destroy data completely.

With RQ3, we address our claim that PISTIS enables stakeholders to both S&M database and retain much of testing utility. RQ1 and RQ2 addressed questions of sanitization and minimization separately w.r.t. the testing efficacy, and this separation allows us to account for confluence and confounding factors. Our novel contribution is in a single approach that both minimizes and sanitizes data and we must ensure that these both functions permit an acceptable testing efficacy.

B. Subject Programs

We evaluate PISTIS using two open-source Java DCAs that belong to different domains and they come with test cases. Our selection of subject programs is influenced by several factors: sizes of the databases, size of the source code, presence of unit, system, and integration tests, and the presence of embedded SQL queries that these programs use. `RiskIt` is an insurance quote program.[3] `DurboDax` enables customer support centers to manage customer data.[4] Table I contains characteristics of the subject DCAs, their databases, and test cases. The first column shows the names of the subject programs, followed by the number of lines of code, LOC for the program code and accompanying test cases. The source code of the projects ranges from 14.2kLOC to 15.7kLOC. Total numbers of statement for tests are 5kLOC and 11kLOC respectively. For the number of test cases, there are 66 from `Durbodax` and 12 from `Riskit`. Other columns show the size of the database, number of tables and attributes in the database, initial test branch coverage with the original database and the number of covered branches. Both databases have more than 88k records.

[3]https://riskitinsurance.svn.sourceforge.net.

[4]http://se547-durbodax.svn.sourceforge.net

DCA	**App**	**Test**	**DB**	**Tbl**	**Att**	**BC**	**NBC**
	[kLOC]		**[KB]**			**%**	
DurboDax	14.2	5.0	791	27	114	19.3	87
RiskIt	15.7	11.0	9681	14	57	13.0	172

TABLE I: Characteristics of the subject DCAs. App = application code, Test = test cases, DB = database, Tbl = tables, Att = attributes in all tables, BC = initial test branch coverage with the original database, NBC = number of covered branches with the original database.

C. Methodology

To evaluate PISTIS, we carry out experiments to explore its effectiveness in enabling users to preserve test coverage while achieving different levels of data privacy and database sizes (RQ1 and RQ2), and to show that it is possible to apply PISTIS to get S&M guarantees (RQ3). All experiments were carried out using Intel Core i5-2520M CPU2.5GHZ with 8GB RAM. The operating system was Windows 7 Professional.

1) Variables: The main independent variable is the value k of clusters, as in k-means clustering. The value k of clusters specifies the data compression rate, since N records in the database will be reduced to k centroid records at the compression rate of $\frac{N}{k}$. Six main dependent variables are the disclosure rate (DR), the number of unique records (UR), the average number of records per cluster (ARPC), branch coverage (BC), and statement coverages (SC). We compute the disclosure rate as the average of all cells in the similarity matrix, an example of which is shown in Figure 5. We measure the number of unique records as the number of centroids that precisely match some original records. Branch coverage is measured as the ratio of the number of covered branches as part of executing test cases to the total number of branches in the DCA.

2) The Structure of the Experiments: For the experiments, we perform program analysis on subject DCAs to obtain attribute rankings, and then we perform weighted k-mean clustering. Unfortunately, data mining algorithms are very computationally intensive especially when running on large datasets. Therefore, we randomly select 2,000 records from each database for each DCA and we apply clustering to these records. Our goal is to run experiments for different values of the independent variable k and report the effect of varying the values on k on dependent variables.

D. Threats to Validity

A threat to the validity of this experimental evaluation is that our subject programs are of moderate size. Large DCAs that have millions of lines of code and use databases whose sizes are measured in thousands of tables and attributes may have different characteristics compared to our small to medium size subject programs. Increasing the size of applications to millions of lines of code may lead to a nonlinear increase in

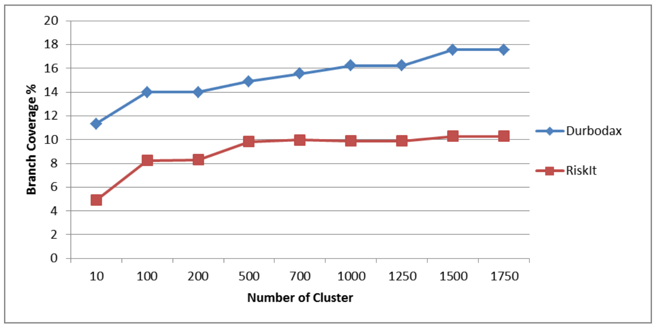

Fig. 7: Dependency of the branch coverage on the number of clusters for subject applications.

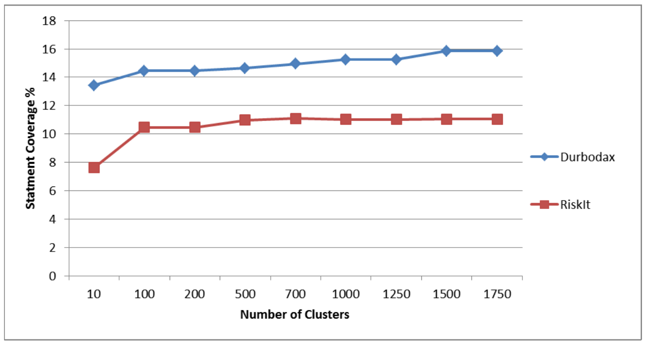

Fig. 8: Dependency of the statement coverage on the number of clusters for subject applications.

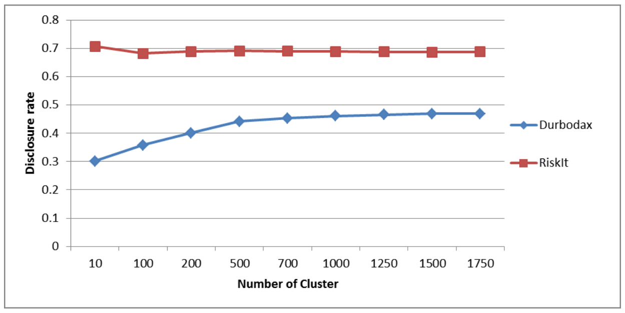

Fig. 9: Dependency of the disclosure rate on the number of clusters for subject applications.

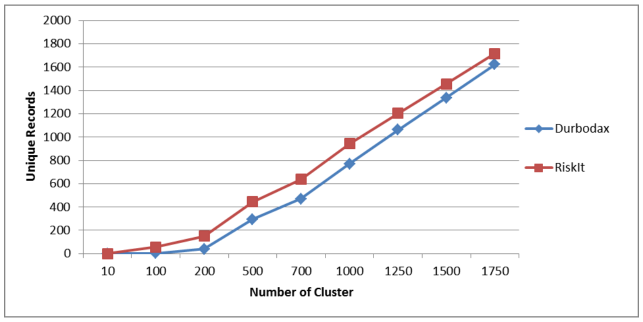

Fig. 10: Dependency of the number of unique records on the number of clusters for subject applications.

the analysis time and space demand for PISTIS. Future work could focus on making PISTIS more scalable.

The other threat to validity is in selecting a subset of records in the database to avoid computational complexity. Indeed, if run with a full set of records, our results could be different. For example, more unique records could be found thereby negatively affecting the disclosure rate. However, since we selected records at random, it mitigates this threat to validity.

In addition, we set high threshold for the support and confidence levels for associative rule mining to achieve two goals: first, we obtain meaningful rules and second, we reduce the computational load that is often directly proportional to the low values of support and confidence. As a result, for `RiskIt` we did not have any associative rules to correct centroid records. The result is, as we believe, the higher number of unique branches that are triggered when running DCAs with S&Med data, which are not triggered with the original data. Although, many of these branches match some of the original branches, further investigation is required to study the effect of associative rules. This is the subject of our future work.

E. Results

The results of the experiments conducted to address *RQs* are shown in Figures 7– 10 and Table II. We compute the centroids based on 4000 random records in the database. For the number of clusters, k, we do interval selection between the numbers from 0 to 2000. We didn't pick $k > 2000$, since the coverage is increase rapidly from k = 10 to 1000, but it's almost flat after $k > 1500$.

The Figure 7 shows dependency of branch coverage on the number of clusters k. The branch coverage increases with a greater number of clusters. The best branch coverages are 17.6% and 10.3% for DCAs `DurboDax` and `RiskIt` respectively. Alternatively, the worst branch coverages are 11.3% and 4.9% respectively. Comparing with the branch coverages of the original datasets, which are 19.33% and 12.99%, the worst case drops in these coverages are only 41.5% and 62.3% when minimizing the dataset from 4,000 to 10 records. In addition, branch coverage of baseline (one record) for `Durbodax` is 8.4% and baseline for `Riskit` is 3.25%. The Figure 8 shows dependency of statement coverage on the number of clusters k. The upward trends are similar with trends in the branch coverage. Drops are 17.5% and 36% for DCAs `DurboDax` and `RiskIt` respectively when minimizing the original datasets to only 10 centroids. In addition, the least drop in test coverage is observed for the DCA `DurboDax`. Our explanation is that `DurboDax` is least sensitive to values of the database attributes since it rarely uses these values in conditional expressions.

The dependency of the disclosure rate from the number of clusters is shown in the Figure 9. For DCA `DurboDax`, lower disclosure rate are observed for smaller numbers of clusters. It means that the centroid records hide more information from the original dataset the more records are put into a cluster. Naturally, it means that more generalization hides more information. However, this statement does not hold true

Subject	Init Stat. Cov %	Init Br. Cov %	Worst St.Cov %	Worst Br.Cov %	Dep Var	The Number of Clusters, k								
						10	100	200	500	700	1000	1250	1500	1750
Durbodax	16.28	19.33	13.42	11.33	DR	0.30	0.36	0.40	0.44	0.45	0.46	0.47	0.47	0.47
					UR	0	3	39	295	469	770	1062	1338	1623
					ARPC	200	20	10	4	2.9	2	1.6	1.3	1.14
					σ	27.25	6.89	3.64	1.70	1.25	0.77	0.44	0.29	0.18
					BC	11.3	14.0	14.0	14.9	15.6	16.2	16.2	17.6	17.6
					SC	13.4	14.4	14.4	14.6	14.9	15.2	15.2	15.9	15.9
RiskIt	11.88	12.99	7.61	4.91	DR	0.707	0.681	0.688	0.691	0.690	0.689	0.687	0.686	0.687
					UR	0	58	149	445	637	942	1203	1457	1716
					ARPC	200	20	10	4	2.9	2	1.6	1.3	1.14
					σ	41.19	17.87	11.72	5.68	3.84	2.55	1.54	0.96	0.48
					BC	4.9	8.2	8.3	9.8	10.0	9.9	9.9	10.3	10.3
					SC	7.61	10.46	10.46	10.97	11.09	11.03	11.03	11.05	11.05

TABLE II: Results of experiments with subject DCAs for different values of the independent variable k that is shown in the last column header that spans nine subcolumns. The second (third resp.) column, Init Stat. Cov (Init Br. Cov resp.), shows the percentage of statement(branch resp.) coverage that is achieved with running test cases against the DCA with the original data in the database. The next two columns show the worst statement/branch test coverage with sanitized and minimized data. The next column lists six dependent variables: the disclosure rate (DR), the number of unique records (UR), the average number of records per cluster (ARPC), the standard deviation for records per cluster, σ, branch coverage in percentage (BC), and statement coverage in percentage (SC). Finally, the last nine subcolumns show values of these dependent variables for different values of k.

for `RiskIt`. Closer investigation of its source code and the database revealed that records in `RiskIt` have same data in many attributes. For example, almost all records in the attribute "BIRTHCOUNTRY" are "United States". Therefore, records in `RiskIt` are more similar with each other and centroids will be much similar with the original data. Since this data is used by the application, it explains that disclosure rate for `RiskIt` is higher than `DurboDax`.

For Figure 10, we compute how many centroids match original records. Naturally, as the number of clusters approaches the number of original records, the number of unique records approaches the number of original records, since each cluster will ultimately have one original record. The figure shows that we have more centroids match original records in `RiskIt` than it in `DurboDax` on same number of clusters. Recall that `RiskIt` have many records that share the same values in the same attribute, and it makes it difficult to effectively hide information by clustering data. Thus, this experiment reveals a limitation of PISTIS: if database records are highly similar to one another, protecting information in this database becomes a very difficult exercise.

Dependencies of branch and statement test coverages on the disclosure rates are shown in Figure 11 and Figure 12 respectively. These graphs are not smooth and monotonic – depending on how centroids are computed, we can see an increase or a decrease in the coverages. Of course, different DCAs can exhibit different dependencies. Our conclusion is that balancing test coverages and disclosure rates is DCA-specific, i.e., stakeholders has to have well-defined impact map of how changing data in the database affects the execution of the DCA.

Generating associative rules took ≈ 1.4 seconds of elapsed time and ≈ 76.7MB of RAM for `DurboDax` and ≈ 10.4 seconds of elapsed time and ≈ 939.5MB of RAM for `RiskIt` using the algorithm APriori [4]. Computing centroids was done using a weighted k-mean algorithm. To compute 10 centroids, it took ≈ 0.7 seconds of elapsed time and ≈ 3.3MB of RAM for `DurboDax` and ≈ 1.7 seconds of elapsed time and ≈ 13.2MB of RAM for `RiskIt`. When increasing the number of centroids to 200, it took ≈ 5.5 seconds of elapsed time and ≈ 15.1MB of RAM for `DurboDax` and ≈ 30.1 seconds of elapsed time and ≈ 88.4MB of RAM for `RiskIt`. Examples of associative rules for `DurboDax` include the following: FARM='All' $\Rightarrow$ YEARR='All' and GQTYPE='(-inf-0.5]' $\Rightarrow$ FARM='All'.

Result summary. These results strongly suggest that PISTIS helps achieve high test coverage for given levels of privacy for subject DCAs, thereby addressing RQ1. PISTIS can also achieve significant reduction in the size of the database while preserving test coverage as it is seen from Table II, thereby addressing RQ2. Finally, the results shown in Figure 7 and 8 strongly suggest that PISTIS is effective in enabling stakeholders to both sanitize and minimize databases while maintaining testing efficacy, thereby addressing RQ3.

V. Related Work

Our work is related to regression testing [47] since PISTIS is used to assess the impact of data anonymization on testing. Numerous techniques have been proposed to automate regression testing. These techniques usually rely on information obtained from the modifications made to the source code. These techniques are not directly applicable to preserving test coverage while achieving data anonymity for test outsourcing, since regression information is derived from changes made to the source code and not to how this code uses databases.

Automatic approaches for test data generation [12], [16], [20], [29], [44] partially address this problem by generating synthetic input data that lead program execution toward untested statements. However, one of the main issues for these approaches is how to generate synthetic input data with which test engineers can achieve good code coverage. Using original data enables different approaches in testing and privacy to produce higher-quality synthetic input data [2] [19, page 42], thus making original data important for test outsourcing.

In *selective anonymization*, where a team is assembled comprising of business analysts and database and security experts [24, page 134]. After the team sets privacy goals, identifies sensitive data, and marks database attributes that may help attackers to reveal this sensitive data (i.e., QIs), anonymization techniques are applied to these QIs to protect sensitive data, resulting in a sanitized database. A goal of all anonymization approaches is to make it impossible to deduce certain facts about entities with high confidence from the anonymized data [3, pages 137-156]. Unfortunately, this approach is subjective, manual, and laborious. In addition, it involves highly trained professionals and therefore this approach is very expensive. Currently, there exists no solution that enables these professionals to accomplish this task efficiently and effectively with metrics that clearly explain the cost and benefits of selective anonymization decisions.

Closely related to PISTIS is $kb-$anonymity model that enables stakeholders to release private data for testing and debugging by combining the $k-$anonymity with the concept of program behavior preservation [8]. Unlike PISTIS, $kb-$anonymity replaces some information in the original data to ensure privacy preservation so that the replaced data can be released to third-party developers. PISTIS and $kb-$anonymity are complementary in using different privacy mechanisms to preserve original data thereby improving its testing utility.

The idea of using clustering for improving k-anonymity was proposed by Bertino et al [9]. A focus of this work is to ensure anonymization of data while at the same time minimizing the information loss resulting from data modifications. Unlike PISTIS, this approach focuses only on applying clustering for k-anonymity and the impact on test coverage and data minimization is not investigated.

Recently proposed is an anonymization technique for protecting private information in bug reports that are delivered to vendors when programs crash on computers of customers [11] and the follow-up work on this technique by Clause and Orso [14]. This technique provides software vendors with new input values that satisfy the conditions required to make the software follow the same execution path until it fails, but are otherwise unrelated with the original inputs. This technique uses symbolic execution to create new inputs that allow vendors to reproduce the bug while revealing less private information than existing techniques. The technique requires test cases, which are not present in our situation. In contrast, PISTIS does not require any test case.

There has been a lot of recent work to achieve general purpose (task-independent) data anonymization. We choose the guessing anonymity approach in this paper because guessing anonymity can be used to provide privacy guarantees for data swapping algorithms and can also provide an optimal noise parameter when implementing data swapping algorithms for anonymization. In contrast, approaches that aim to achieve

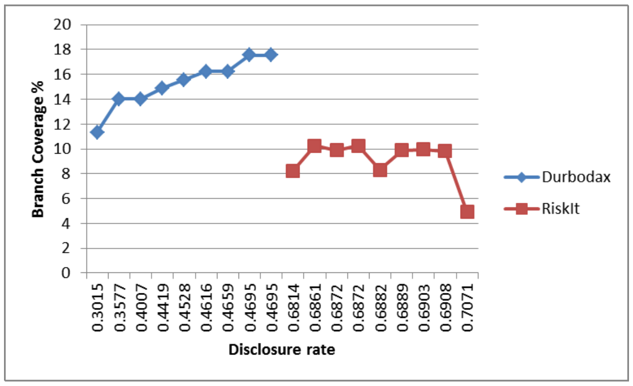

Fig. 11: Branch coverage as a function of the disclosure.

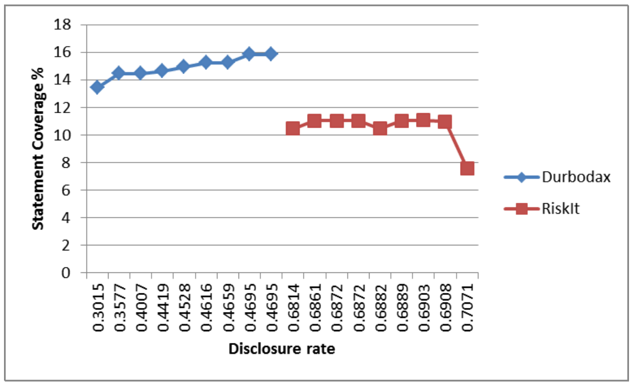

Fig. 12: Statement coverage as a function of the disclosure.

k-anonymity do not allow the user to explicitly control how much each record is altered. Empirical results reported by Rachlin et al. [38] show that Guessing anonymity outperforms DataFly, a well-known k-Anonymity algorithm on specific data mining tasks, namely classification and regression, while at the same time providing a higher degree of control over how much the data is distorted.

Recent work on privacy introduced a similar definition of privacy for noise perturbation methods, known as k-randomization [1]. This work defines a record as k-randomized if the number of records that are a more likely match to the original is *at least k*. Although this notion is similar to the definition of guessing anonymity, the definition differs by not providing a lower limit on the number of records that provide a more likely match, and by explicitly establishing a connection between privacy and guessing functions.

VI. Conclusion and Future Work

We propose a novel approach for *Protecting and mInimizing databases for Software TestIng taSks (PISTIS)* that both sanitizes a database and minimizes it. PISTIS uses a weight-based data clustering algorithm that partitions data in the database using information from program analysis that indicate how this data is used by the application. For each cluster, a centroid object is computed that represents different persons or entities in the cluster, and we use associative rule mining to compute and use constraints to ensure that the centroid objects are representative of the general population of data in the cluster. Doing so also sanitizes information, since these centroid objects replace the original data to make it difficult for attackers to infer sensitive information. Thus, we reduce a large database to a few centroid objects and we show in our experiments with four applications that test coverage stays within a close range to its original level.

Our experimental results are promising, and there are several areas that will improve our work. First, we plan to adapt PISTIS to other test coverage metrics, such as path coverage and MC/DC coverage, to study if PISTIS can generalize to other metrics. Next, we plan to investigate the relationship between disclosure rates and fault-detection abilities with PISTIS. Since there is a body of research that shows a strong correlation between improved test coverage and fault location [10], [27], [34], [37], we expect that using PISTIS increases the probability of finding faults while sanitizing and minimizing data. Finally, we plan to improve the implementation of PISTIS to reduce the analysis time. We identified several bottlenecks from our experiments, i.e., data mining approach are computationally intensive. With more engineering on the bottlenecks, specifically by using massive parallelization platform like MapReduce, we expect PISTIS to run faster and thus be applicable in many real-world scenarios.

VII. Acknowledgments

We thank Daniel Graham from the College of William and Mary for his contributions to early versions of this work and Qi Luo from W&M for helping us to reproduce the results of the experiments presented in the paper. This material is based upon work supported by the NSF under Grants No. CCF-1218129, CCF-1016868, CCF-1017633, CCF-1217928 and Microsoft SEIF award. Any opinions, findings and conclusions expressed in this work are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] C. C. Aggarwal. On randomization, public information and the curse of dimensionality. In *ICDE*, pages 136–145, 2007.

[2] C. C. Aggarwal and P. S. Yu. On static and dynamic methods for condensation-based privacy-preserving data mining. *TODS*, 2008.

[3] C. C. Aggarwal and P. S. Yu. *Privacy-Preserving Data Mining: Models and Algorithms*. Springer, 2008.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.

[5] W. Aspray, F. Mayades, and M. Vardi. *Globalization and Offshoring of Software*. ACM, 2006.

[6] C. Bizer, P. Boncz, M. L. Brodie, and O. Erling. The meaningful use of big data: four perspectives – four challenges. *SIGMOD Rec.*, 40(4):56–60, Jan. 2012.

[7] V. R. Borkar, M. J. Carey, and C. Li. Big data platforms: what's next? *XRDS*, 19(1):44–49, Sept. 2012.

[8] A. Budi, D. Lo, L. Jiang, and Lucia. *b*-anonymity: a model for anonymized behaviour-preserving test and debugging data. In *PLDI*, pages 447–457, 2011.

[9] J.-W. Byun, A. Kamra, E. Bertino, and N. Li. Efficient *k* -anonymization using clustering techniques. In *DASFAA*, pages 188–200, 2007.

[10] X. Cai and M. R. Lyu. The effect of code coverage on fault detection under different testing. In *A-MOST*, pages 1–7, 2005.

[11] M. Castro, M. Costa, and J.-P. Martin. Better bug reporting with better privacy. In *ASPLOS*, pages 319–328, 2008.

[12] L. A. Clarke. A system to generate test data and symbolically execute programs. *IEEE Trans. Softw. Eng.*, 2(3):215–222, 1976.

[13] J. Clause and A. Orso. Penumbra: Automatically identifying failure-relevant inputs using dynamic tainting. In *ISSTA*, pages 249–260, 2009.

[14] J. A. Clause and A. Orso. Camouflage: automated anonymization of field data. In *ICSE*, pages 21–30, 2011.

[15] Datamonitor. Application testing services: global market forecast model. *Datamonitor Research Store*, Aug. 2007.

[16] R. A. DeMillo and A. J. Offutt. Constraint-based automatic test data generation. *IEEE Trans. Softw. Eng.*, 17(9):900–910, 1991.

[17] L. K. Dillon and Q. Yu. Oracles for checking temporal properties of concurrent systems. In *ACM SIGSOFT*, pages 140–153, Dec. 1994.

[18] M. Emmi, R. Majumdar, and K. Sen. Dynamic test input generation for database applications. In *ISSTA*, pages 151–162, 2007.

[19] B. C. M. Fung, K. Wang, A. W.-C. Fu, and P. S. Yu. *Introduction to Privacy-Preserving Data Publishing: Concepts and Techniques*. Data Mining and Knowledge Discovery. Chapman & Hall/CRC, August 2010.

[20] P. Godefroid. Compositional dynamic test generation. In *POPL*, pages 47–54, 2007.

[21] M. Grechanik, C. Csallner, C. Fu, and Q. Xie. Is data privacy always good for software testing? In *ISSRE*, pages 368–377, 2010.

[22] F. Haftmann, D. Kossmann, and E. Lo. A framework for efficient regression tests on database applications. *The VLDB Journal*, 16(1):145–164, Jan. 2007.

[23] J. Horey, E. Begoli, R. Gunasekaran, S.-H. Lim, and J. Nutaro. Big data platforms as a service: challenges and approach. In *USENIX*, pages 16–16, 2012.

[24] C. Jones. *Software Engineering Best Practices*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2010.

[25] C. Kaner, J. Bach, and B. Pettichord. *Lessons Learned in Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[26] G. M. Kapfhammer and M. L. Soffa. A family of test adequacy criteria for database-driven applications. In *ESEC/FSE*, pages 98–107, 2003.

[27] Y. W. Kim. Efficient use of code coverage in large-scale software development. In *CASCON*, pages 145–155, 2003.

[28] A. Labrinidis and H. V. Jagadish. Challenges and opportunities with big data. *Proc. VLDB Endow.*, 5(12):2032–2033, Aug. 2012.

[29] R. Majumdar and K. Sen. Hybrid concolic testing. In *ICSE*, pages 416–426, 2007.

[30] S. Mizzaro. Relevance: The whole history. *JASIS*, 48(9):810–832, 1997.

[31] S. Mizzaro. How many relevances in information retrieval? *Interacting with Computers*, 10(3):303–320, 1998.

[32] S. S. Muchnick. *Advanced compiler design and implementation*. Morgan Kaufmann, 1997.

[33] T. E. Murphy. Managing test data for maximum productivity. *http://www.gartner.com/DisplayDocument ?doc_cd=163662&ref=g_economy_2reduce*, Dec. 2008.

[34] A. S. Namin and J. H. Andrews. The influence of size and coverage on test suite effectiveness. In *ISSTA*, pages 57–68, 2009.

[35] D. Peters and D. L. Parnas. Generating a test oracle from program documentation: work in progress. In *ISSTA*, pages 58–65, 1994.

[36] F. Peters and T. Menzies. Privacy and utility for defect prediction: Experiments with morph. In *ICSE*, pages 189–199, 2012.

[37] P. Piwowarski, M. Ohba, and J. Caruso. Coverage measurement experience during function test. In *ICSE*, pages 287–301, May 1993.

[38] Y. Rachlin, K. Probst, and R. Ghani. Maximizing privacy under data distortion constraints in noise perturbation methods. In *PinKDD*, 2008.

[39] D. J. Richardson. Taos: Testing with analysis and oracle support. In *ISSTA*, pages 138–153, 1994.

[40] D. J. Richardson, S. Leif-Aha, and T. O. O'Malley. Specification-based Test Oracles for Reactive Systems. In *ICSE*, pages 105–118, 1992.

[41] I. Shield. International data privacy laws. *http://www.informationshield.com/intprivacylaws.html*, 2010.

[42] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[43] K. Taneja, M. Grechanik, R. Ghani, and T. Xie. Testing software in age of data privacy: a balancing act. In *FSE*, pages 201–211, 2011.

[44] K. Taneja, Y. Zhang, and T. Xie. MODA: Automated test generation for database applications via mock objects. In *ASE*, pages 289–292, 2010.

[45] B. G. Thompson. *H.R.6423: Homeland Security Cyber and Physical Infrastructure Protection Act of 2010*. U.S.House, 2010.

[46] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[47] S. Yoo and M. Harman. Regression testing minimisation, selection and prioritisation: A survey. *STVR*, 2011.