Testing Software In Age Of Data Privacy: A Balancing Act

Kunal Taneja
North Carolina State University
Raleigh, NC 27695
ktaneja@ncsu.edu

Mark Grechanik
Accenture Technology Labs
Chicago, IL 60601
mark.grechanik@accenture.com

Rayid Ghani
Accenture Technology Labs
Chicago, IL 60601
rayid.ghani@accenture.com

Tao Xie
North Carolina State University
Raleigh, NC 27695
xie@csc.ncsu.edu

ABSTRACT

Database-centric applications (DCAs) are common in enterprise computing, and they use nontrivial databases. Testing of DCAs is increasingly outsourced to test centers in order to achieve lower cost and higher quality. When proprietary DCAs are released, their databases should also be made available to test engineers. However, different data privacy laws prevent organizations from sharing this data with test centers because databases contain sensitive information. Currently, testing is performed with anonymized data, which often leads to worse test coverage (such as code coverage) and fewer uncovered faults, thereby reducing the quality of DCAs and obliterating benefits of test outsourcing.

To address this issue, we offer a novel approach that combines program analysis with a new data privacy framework that we design to address constraints of software testing. With our approach, organizations can balance the level of privacy with needs of testing. We have built a tool for our approach and applied it to nontrivial Java DCAs. Our results show that test coverage can be preserved at a higher level by anonymizing data based on their effect on corresponding DCAs.

Categories and Subject Descriptors

D.2.5 [**Software Engineering, Testing and Debugging**]: Testing tools; D.4.6 [**Software Engineering, Security and Protection**]: Information flow controls; K.4.1 [**Computers and Society, Public Policy Issues**]: Privacy

General Terms

Security, Verification

Keywords

Data anonymity, software testing, privacy framework, utility, PRIEST

1. INTRODUCTION

Large organizations today face many challenges when engineering software applications. Particularly challenging is the fact that many applications work with existing databases that contain confidential data. A large organization, such as a bank, insurance company, or government agency, typically hires an external company to develop or test a new custom software application. However, recent data protection laws and regulations [35] around the world prohibit data owners to easily share confidential data with external software service providers.

Database-centric applications (DCAs) are common in enterprise computing, and they use nontrivial databases [26]. When releasing these proprietary DCAs to external test centers, it is desirable for DCA owners to make their databases available to test engineers, so that they can perform testing using *original data*. However, since sensitive information cannot be disclosed to external organizations, testing is often performed with *synthetic input data*. For instance, if values of the field *Nationality* are replaced with the generic value "`Human`," DCAs may execute some paths that result in exceptions or miss certain paths [23]. As a result, test centers report worse test coverage (such as code coverage) and fewer uncovered faults, thereby reducing the quality of applications and obliterating benefits of test outsourcing [30].

Automatic approaches for test data generation [12, 17, 22, 29, 37] 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 [3][21, page 42], thus making original data important for test outsourcing.

A fundamental problem in test outsourcing is how to allow a DCA owner to release 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 data (sanitized data or anonymized data is the original data after anonymization. We use the two terms interchangeably throughout this paper) should induce execution paths that are similar to the ones that are induced by the original data. In other words, when data is sanitized, information about how DCAs use this data should be taken into consideration. In practice, this consideration rarely happens; our previous work [23] showed that a popular data anonymization algorithm, called *k*–anonymity, seriously degrades test coverage of DCAs.

Naturally, different DCAs have different privacy goals and levels of data sensitivity – privacy goals are more relaxed for a DCA that manages a movie ticket database than for a DCA that is used within banks or government security agencies. Applying more relaxed protection to databases is likely to result in greater test coverage since a small part of the databases is anonymized; conversely,

ESEC/FSE'11, September 5–9, 2011, Szeged, Hungary.
Copyright 2011 ACM 978-1-4503-0443-6/11/09 ...$10.00.

stricter protection makes it more difficult to outsource testing. The latter is the result of two conflicting goals: making testing as realistic as possible and hiding the original data from testers who need this data to make testing effective. Balancing these goals, i.e., to anonymize data while preserving test coverage of DCAs that use this data is emerging to be an important problem.

Given the importance of this problem, it may be surprising that there exists little prior research on this topic. There may be two main reasons for the lack of prior research. First, elaborate data privacy laws are a new phenomenon, and many of these laws [35, 38] have been introduced after the year 2000. Second, it is only in the past decade that applications are increasingly being tested by third-party specialized software service providers, which are also called test centers. Numerous test centers have emerged and often offer lower cost and higher quality when compared to in-house testing. In 2007, the test outsourcing market was worth more than USD 25 billion and growing at 20% annually, making test outsourcing the fastest growing segment of the application services market [7, 16].

To address this issue, we offer a novel approach, *PRIvacy Equalizer for Software Testing (PRIEST)* that combines a new data privacy framework with program analysis enabling business analysts to determine how anonymizing values of database attributes affects test coverage. With PRIEST, organizations can balance the goals of preserving test coverage, while releasing DCAs to external test centers with a controlled disclosure of sensitive information. The source code for PRIEST as well as its illustration video are publicly available[1]. Our work is unique; to the best of our knowledge, there exists no prior approach that addresses the problem that we pose in this paper. In summary, we make the following main contributions in this paper:

- We create a new privacy framework (described in Section 4) that includes a novel combination of guessing anonymity-based privacy metrics and a technique of data swapping anonymization to enable organizations to keep original values in sanitized data; keeping such values is important for improving the effectiveness of testing.

- We design and implement a technique using program analysis for determining how values of database attributes affect test coverage of DCAs that use this data (see Section 3.3).

- We combine our privacy framework with this technique in PRIEST to enable business analysts to balance data privacy with test coverage.

- We evaluate PRIEST using three open-source Java DCAs and one large Java DCA that handles logistics of one of the largest supermarket chains in Spain. We show that with PRIEST, test coverage can be preserved at a higher level by pinpointing database attributes that should be anonymized based on their effect on corresponding DCAs.

2. THE PROBLEM

In this section, we provide the necessary background on how DCAs interact with databases, show how sanitizing data affects test coverage of DCAs, describe the state of the art and practice, and formulate the problem statement.

2.1 Background

Majority of enterprise-level DCAs use programs that are written in general-purpose programming languages and relational databases to maintain large amounts of data. A primary way for these programs to communicate with databases is to use *call-level interfaces*, which allow DCAs to access database engines through standardized *application programming interfaces (APIs)*, e.g., Java DataBase Connectivity (JDBC). Using JDBC, programs pass SQL queries as strings to corresponding API calls to be sent to databases for execution.

```
recordset = db.execSql("SELECT␣*␣FROM␣TblPatient");
nationality = recordset.GetAttribute( i );
age = recordset.GetAttribute( j );
disease = = recordset.GetAttribute( "Disease" );
if (nationality=="Palauan" && age>60) f(disease);
```

Figure 1: An illustrative example that shows how replacing values of database attributes `Nationality` and `Age` of patients can make the function `f` unreachable.

Once these queries are executed, values of attributes of database tables are returned to DCAs using JDBC `RecordSet` objects and these values are stored in variables of the DCAs, which in turn use these variables as part of their application logic. In this way, values from a database may be used in branch decisions, and these values may therefore affect the subsequent execution of the DCAs. Depending on returned values, different paths can be taken in DCAs, and subsequently these values affect test coverage. Hence removing certain classes of values in database attributes may make some branches and statements in DCAs unreachable.

To see how anonymization affects test coverage of DCAs, consider a fragment of code shown in Figure 1. After executing JDBC API calls that submit an SQL query to a database and obtain the object `recordset`, the values of the attributes `Age`, `Nationality`, and `Disease` are put in the corresponding variables of the DCA `nationality`, `age`, and `disease`, respectively. If the values of the attribute nationality are replaced with the generic value "`Human`,", the function call `f()` becomes unreachable.

Certain classes of values of database attributes that contain non-sensitive information should be anonymized. These attributes, also called *quasi-identifiers (QI)* often contain information that can be linked with other data to infer sensitive information about entities (i.e., people, objects). For example, given the values of the QIs `Race`, `Sex`, `Height`, `ZipCode` and the attribute that contains sensitive data about diseases, it is possible to link a person to specific diseases, provided that the values of these QIs uniquely identify this person.

Existing data anonymization approaches are centered on creating models for privacy goals and developing algorithms for achieving these goals using particular anonymization techniques [15]. One of the most popular privacy goals is $k-$anonymity [34], where each entity in the database must be indistinguishable from $k-1$ others. Anonymization approaches use different anonymization techniques including suppression, where information (e.g., nationality) is removed from the data, and generalization, where information (e.g., age) is coarsened into sets (e.g., into age ranges) [15]. These and other techniques modify or suppress values of attributes, and a common side-effect of these modifications is non-covered statements in DCAs that are otherwise covered with the original data.

2.2 State of the Art and Practice

After interviewing professionals at IBM, Accenture, two large health insurance companies, a biopharmaceutical company, two large supermarket chains, and three major banks, we found that current test data anonymization is manual, laborious, and error-

[1] http://www.privacytesting.org

prone. In a few cases, client companies outsource testing using an especially expensive and cumbersome testing procedure called *cleanroom testing*, where DCAs and databases are kept on company premises in a physically secured environment [23]. Typically, business analysts and test managers from outsourcing companies come to the cleanrooms of their client companies to evaluate their clients' DCAs and to plan work. However, when better options to access data are not available, testers from outsourcing companies are also allowed in these cleanrooms to test software on their clients' premises. Actions of these test engineers are tightly monitored; network connections, phone calls, cameras, and USB keys are forbidden. Cleanroom testing requires significant resources and physical proximity of test outsourcing companies to their clients.

A more commonly used approach is to use tools that anonymize databases indiscriminately, by generalizing or suppressing all data. Even though this procedure is computationally intensive, it is appealing since it does not require sophisticated reasoning about privacy goals and protects all data.

But in many real-world settings, protecting all data blindly makes testing very difficult. When large databases are repopulated with fake data, it is likely that many implicit dependencies and patterns among data elements are missing, thereby reducing testing efficacy. Moreover, fake data is likely to trigger exceptions in DCAs, leading test engineers to flood bug-tracking systems with false bug reports. In addition, testers often cannot use such anonymized data because DCAs may throw exceptions that would not occur when the DCAs are tested with original data or other real data in field.

A more sophisticated approach is *selective anonymization*, where a team is assembled comprising of business analysts and database and security experts [25, 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 [4, 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.

2.3 Balancing Utility And Privacy

Utility of anonymized data is measured in terms of usefulness of this data for computational tasks when compared with how useful the original data is for the same tasks. Consider an example of the utility of calculating average salaries of employees. Adding random noise to protect salary information will most likely result in computing incorrect values of average salaries, thereby destroying utility of this computation. Recent cases with U.S. Census show that applying data privacy leads to incorrect results [5, 8]. Multiple studies demonstrate that even modest privacy gains require almost complete destruction of the data-mining utility [1, 9, 18, 19, 28].

Data swapping is an anonymization technique that is based on exchanging values of attributes among individual records while maintaining certain distribution properties [33, 32]. Data swapping is more effective in preserving utility than data suppression and generalization privacy algorithms since data swapping allows users to better preserve statistical information [21, page 42]. In this paper, we develop a data swapping privacy algorithm that allows analysts to balance privacy and utility goals, i.e., to choose appropriate levels of privacy that will guarantee certain basic test coverage.

2.4 The Problem Statement

The problem statement that we address in this paper is how to enable stakeholders to balance a level of test coverage for DCAs (i.e., utility) with privacy for databases that these DCAs use. The problem space is restricted by three fundamental constraints of software development and privacy-preserving data publishing as follows.

First, a chosen anonymization approach should preserve original data as much as possible since it is important for preserving the testing utility of DCAs. Anonymizing databases using data suppression techniques may result in a different behavior of the DCAs. Suppose that $N_o \subseteq N$ is the set of all covered nodes in the control flow graph of a DCA when the DCA is tested with the original data, and $N_a \subseteq N$ is the set of all covered nodes when the same DCA is tested with the anonymized data. In general, testing with anonymized data makes DCAs behave in ways that are different from specifications, and as a result new execution paths in DCAs with anonymized data may lead to exceptions or not-covered branches being covered originally. For example, the DCA logic handles suppressed values of `Nationality` by not covering the body of the `if` statement shown in Figure 1. Therefore, in the worst case $N_T = N_o \cap N_a = \emptyset$, where N_T is the set of nodes of the preserved statement coverage. A problem is how to keep a good extent of the original data in databases for testing DCAs while guaranteeing certain levels of privacy.

Our goal is to ensure that all statements (i.e., nodes in the control flow graph) that are executed with original data will also be executed with anonymized data. Our goal is difficult to achieve since it is an undecidable problem to determine precisely how values of QIs are used in DCAs [27]. In addition, anonymization algorithms present a dilemma: suppressing attribute data with different values results in loss of test coverage, and keeping original data in the database results in loss of privacy. Balancing these conflicting outcomes is the problem that we address in this paper.

Second, as a result of software evolution [20, page 163], the code of the DCA is modified. In consecutive releases, the DCA may use different database attributes in different ways, thereby requiring the DCA owner to reanonymize data to ensure that the balance between privacy and utility is maintained. However, re-anonymization introduces a problem – if an attacker keeps the previous version of the anonymized data where values of some attributes are not sanitized, then this attacker can link original values in different releases, thus inferring sensitive information. A solution should enable stakeholders to repeatedly release anonymized data in such a way that both privacy and utility are guaranteed at certain levels.

Finally, a privacy metric should be linked directly to test coverage and vice versa, i.e., guaranteeing a certain level of test coverage should allow stakeholders to calculate bounds of the privacy level. For example, if some path is controlled by branch conditions that use values of database attributes, then it is possible to predict the effect of anonymization of these values on this path. In other words, the problem is to present business analysts with choices of predicted coverage levels for different anonymization goals.

3. OUR SOLUTION

In this section, we present core ideas behind our approach that we call *PRIvacy Equalizer for Software Testing (PRIEST)* and we describe the PRIEST architecture and its workflow. In this section, we concentrate on the overall architecture of PRIEST that uses our privacy framework that we describe in Section 4.

3.1 Core Ideas

At the core of our work are three major ideas. The first one is a privacy framework that enables organizations to keep original values in sanitized data. The framework is based on a data swapping anonymization technique to preserve original values of database attributes. As a result, test coverage is not affected so negatively as it happens when data suppression and generalization techniques are used. The technique swaps data based on a probability value provided by a user. Hence different levels of privacy can be achieved.

The second idea is our guessing anonymity privacy metric that allows stakeholders to quantify the level of privacy achieved in an anonymized database. In particular, the metric provides measurement of difficulty for an attacker to relate a sanitized record to the original record.

The third idea is an idea to 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, business analysts and security experts can determine what anonymization strategy should be used to protect data without sacrificing much of test coverage. A key point is that often not all of the database attributes have to be anonymized to achieve a given level of data protection. Therefore, it is important to extend data protection strategies with information about how DCAs use their databases to which these strategies are applied.

As it often happens, control-flow decisions in DCAs are affected by a smaller subset of database attributes. In an extreme case, if some data in the database is not used in any control-flow decision of any DCAs, then anonymizing this data will have no effect on these DCAs. A more subtle point is that values of some attribute may not affect most branch conditions in DCAs, and therefore test coverage will not be affected much if this attribute is anonymized. Thus it is beneficial to focus anonymization on those aspects of the data that have minimal influence on deeply nested control-flow decisions.

3.2 PRIEST Architecture and Process

We propose a novel process for using PRIEST that partially involves the cleanroom testing that we described in Section 2.2. Recall that business analysts and test managers from outsourcing companies come to the cleanroom of their client company to evaluate their client's DCAs and to plan work. As part of their evaluation, these analysts and managers determine different sets of attributes of the database (i.e., QIs) that can be used by attackers to re-identify entities. In general, only a few subsets of these QIs should be anonymized, thus creating favorable conditions for preserving test coverage. With PRIEST, this QI selection procedure can be improved by pointing out the QIs that affect test coverage the least. These analysts and managers can use PRIEST as part of their evaluation and planning in order to determine how to maximize test coverage for DCAs while achieving desired privacy goals for databases that these DCAs use.

As the first step of the PRIEST process, programmers link program variables to database attributes using annotations, so that these annotations can be traced statically using control- and dataflow 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. Our goal is to quantify the effect of replacing values of database attributes on reachability of program statements.

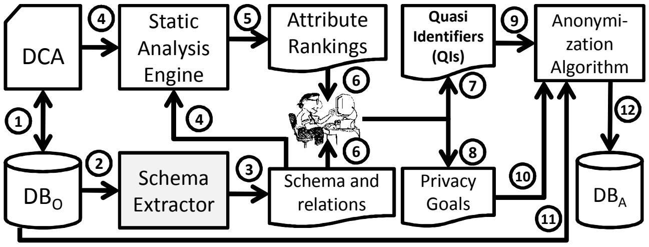


Figure 2: PRIEST architecture and workflow. Solid arrows depict the flow of command and data between components, numbers indicate the sequence of operations in the workflow.

At first glance, it appears to be tedious and laborious work for programmers to annotate program variables with the names of database attributes from which these variables obtain values. In reality, it is a practical and modest exercise that takes little time. Programmers annotate selected program variables only once, where these variables are first assigned values that are retrieved from recordset objects using specific database-related API calls. We observe that in many projects it is a small fraction of code that deals with obtaining values from databases, and most code is written to implement application logic that processes these values. This observation is confirmed by our previous study [24] that shows that out of 2,080 randomly chosen Java programs in Sourceforge, there is approximately one JDBC-related API call per 2,200 LOC on average that retrieves a value from a recordset object and assigns it to a program variable. Extrapolating this result means that programmers may have to annotate approximately 450 variables for a project with 1 Million LOC and such expense is acceptable.

In addition, a variety of *object-relational mapper (O/R mapping)* tools and frameworks bridge the gap between an application's object model and the relational schema of the underlying database by generating classes that represent objects in relational databases [6]. Since O/R mapping is done automatically, links between program variables and database attributes are recovered as a by-product of using O/R mapping tools. For example, one of our subject applications, a logistics application for handling one of the largest supermarket chains in Spain, is written using iBatis[2], an open-source O/R mapper.

Figure 2 shows the architecture of PRIEST. The inputs to PRIEST are the DCA bytecode and the original database DB_O that this DCA uses (1). With PRIEST, business analysts, test managers, and security experts connect to DB_O (2) to obtain its schema using the Schema Extractor that uses JDBC metadata services to obtain (3) database schema including all relations among different attributes. Next, PRIEST performs control and dataflow analyses (4) using the Soot toolkit[3] 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 statement coverage) should not be picked as QIs. The output (5) 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.

At this point, PRIEST displays to the user (6) the list of ranked

[2] http://ibatis.apache.org

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

attributes and their relations with other attributes in the database schema. The user selects (7) a subset of database attributes as QIs whose values should be anonymized to protect sensitive data based on certain privacy goals (8) that are required by the DCA owner. The selected QIs are supplied (9) to the Anonymization Algorithm along with (10) the required privacy level. In addition, the algorithm takes (11) DB_O as its input and outputs (12) the anonymized database DB_A.

3.3 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 taint analysis to track the annotated variables (as described in Section 3.2) corresponding to each attribute. 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. We then count the number of statements that are control-dependent on these branch conditions. We perform virtual-call resolution using static class hierarchy analysis, and we take a conservative approach by counting all the statements in all the target methods that can potentially be invoked.

Our experiments on subject applications show that this approach can predict the effect of anonymization on coverage within less than 8.3% error (see Section 5 and Table 6 for further details). Improving the precision to compute how many statements are affected by these attributes is a subject of future work.

4. PRIEST PRIVACY FRAMEWORK

In this section, we describe the PRIEST privacy framework. We discuss constraints, show how to connect different aspects of data privacy with the testing utility, and present a privacy metric and our anonymization algorithm.

4.1 Constraints and Goals

After discussions with a number of experts from different organizations who are involved in testing DCAs, we identified two main constraints for a successful anonymization solution: simplicity and consistency. A simple anonymization solution should not impose significant laborious manual or intellectual effort on stakeholders who are involved in protecting privacy. A case at hand is when using data suppression and generalization algorithms to achieve *k*-anonymity, a popular data privacy approach, users must specify generalization hierarchies that guide corresponding anonymization algorithms to replace data values with generalized substitutes to protect sensitive information. For example, a branch of the generalization hierarchy for the attribute Nationality could be $\{Canadian, Mexican, USA\} \rightarrow NorthAmerican \rightarrow American \rightarrow Human$. In general, identifying and properly using these hierarchies involves deep domain expertise and significant effort. Stakeholders want a high degree of automation where they are presented with choices for privacy levels and corresponding test coverages, making it easy for them to balance privacy and utility.

Consistency of data is the other constraint. Since data used in typical DCAs can span multiple tables in databases, any field that is used as a key to link entities across tables needs to be anonymized *consistently* so that the data can be linked correctly after anonymization. Thus, anonymization techniques should take into consideration different constraints among attributes that are imposed by the database schema for describing this data.

Preserving original data values is a goal of our anonymization framework – no new values for a data field should be introduced and the unique set of values in a field should be preserved after anonymization. This constraint is important because changing the values (such as generalizing a five digit zip code to first three digits, or replacing the city name with the state in the City field would result in new values for a field that the DCA may not be able to handle properly. Having new data values often results in extra effort to modify the DCA and may cause unintended consequences in production with real data; for example, exceptions that are thrown would not be thrown if the original data is used.

Random data generation would typically preserve only univariate properties (marginal distributions) of each attribute. Our approach can preserve distributions over conjunctions of attributes (Gender=Female, Diagnosis=Ovarian Cancer) that are useful for preserving test coverage when conditional statements containing variables linked to multiple attributes are concerned, and such cases are common in real-world DCAs.

Finally, in this paper, we assume that the developer or data owner cannot be an attacker. While other attack models are possible, we assume that since developers have direct access to all information that is needed to create software, it is reasonable to assume that developers enjoy a high level of trust.

4.2 Guessing Anonymity

Guessing anonymity [31] is a privacy metric that enables stakeholders to quantify the level of privacy using a guessing strategy of the attacker as a sequence of questions of the form “Are these the original values of quasi-identifiers that are used to generate a sanitized record?”

Definition: The guessing anonymity of the sanitized record is the number of guesses that the optimal guessing strategy of the attacker requires in order to correctly guess the record used to generate the sanitized record.

To illustrate this definition, consider an attacker with knowledge that Alice and Chris are in the database, and knowledge of their true ages shown in Table 1 that contains original data. Sanitized data is shown in Table 2 where names are replaced with “***” and the values of the attribute Age are perturbed with random noise. The guessing anonymity of each record is shown in the fourth column of Table 2. While the record that corresponds to Alice has a guessing anonymity of two, the sanitized record corresponding to Chris has a guessing anonymity of one due to the fact that his age is significantly higher than the ages in the other records. The distribution of guessing anonymity of the different records in a database can be used to define a variety of privacy metrics. *Mean guessing anonymity*, *Minimum guessing anonymity*, and *Fraction of records* with guessing anonymity greater than a certain value *m* are some metrics that we discuss later in this section. For our toy database shown in Table 1, those values would be 1.75, 1, and 0.75 (for *m*=1), respectively.

To further illustrate our definition of guessing anonymity, we link our definition to the definition of *k*-anonymity. Recall that in *k*-anonymity, a database is considered private if every record has at least *k* other records in the original database with which the released record is indistinguishable. *k*-anonymity can be achieved by different anonymization operators but typically suppression and generalization operators are the most commonly used ones.

Consider a released record that is anonymized such that there are exactly *k* records in the original database with which the released record is consistent. Without any further information, the optimal guessing strategy would choose among these *k* records with uniform probability for its first guess. The probability of the first

Table 1: Original Database

Name	Age	Procedure or Prescription
Alice	19	Antidepressants
Bob	15	Antibiotics
Chris	52	Chemotherapy
Diana	25	Abortion

Table 2: Sanitized Database

Name	Age	Procedure or Prescription	Guessing Anonymity
***	23.1	Antidepressants	2
***	19.4	Antibiotics	2
***	49.3	Chemotherapy	1
***	21.1	Abortion	2

Algorithm 1 The `PriestPrivacy` algorithm.

```
1: PriestPrivacy( $\{QI\}$, $p$ ) $\{\{QI\}$ is the set of QIs, and $p$ is the
   probability that the original data will remain unchanged in the
   anonymized set, $T\}$
2: $\|T\| \leftarrow$NULL {Initialize values of the anonymized matrix.}
3: for $j \leftarrow 1$ to # of attributes in $\|QI\|$ do
4:     DistinctValues($\|QI\|_j$) $\mapsto \{V\}_j$ {Returns the set of distinct
       values for a given attribute.}
5:     for $i \leftarrow 1$ to # of rows in $\|QI\|$ do
6:         Randomize($QI^i_j, p$) $\mapsto (\exists v \in \{V\}_j$ s.t. $T^i_j \leftarrow v)$
7:     end for
8: end for
9: return $\|T\|$
```

guess being correct is $\frac{1}{k}$. If the first guess is incorrect, the second guess is chosen with uniform probability from among the remaining $k-1$ records. The probability of the second guess being correct is $\frac{1}{k-1}(1-\frac{1}{k})$. The expected number of guesses simplifies to $\frac{k+1}{2}$, so the expected guessing anonymity of a k-anonymized record is $\frac{k+1}{2}$.

4.3 The PRIEST Anonymization Algorithm

The algorithm `PriestPrivacy` is shown in Algorithm 1. We use a data swapping anonymization technique to preserve original values of database attributes. The goal is to preserve test coverage better than when data suppression and generalization techniques are used, while keeping the data usable by the DCA. To protect privacy, each value for each row for each attribute is swapped with some probability with a different value for the same attribute. For example, for the attribute `Gender` that contains two distinct values `M` and `F`, the user may choose to replace the value of a given cell with the other value with probability 0.5, i.e., an unbiased coin flip. Excluding an attribute from anonymization means that the probability of value replacement for this attribute is zero.

This algorithm takes as its inputs the matrix of QIs and their values, $\|QI\|$, whose columns include QIs and rows include different tuples for these QIs from the original database DB_O, and the value of the probability, p, that the original data will remain unchanged in the anonymized matrix, $\|T\|$. The matrix $\|T\|$ has the same dimensions and semantics as the matrix $\|QI\|$, and it contains anonymized values of QIs. $\|T\|$'s values are initialized to `NULL` in Line `2` and this matrix is returned in Line `9`.

The `for`-loop in Lines `3-8` iterates through attributes that are in $\|QI\|$, and the procedure `DistinctValues` is called to compute the set of distinct values for each QI, $\{V\}$. Next, in Lines `5-7` the nested `for`-loop iterates through all rows for the given QI and the procedure `Randomized` is invoked to replace the original value in a cell with one of the original distinct values of the given QI, and the replaced value is written in the corresponding location in the matrix $\|T\|$. Once the algorithm iterates over all QIs and all rows for each QI, it terminates and returns $\|T\|$ in Line `9`.

Algorithm 2 Guessing anonymity metric calculation algorithm.

```
PrivacyMetric( $\|QI\|$, $\|T\|$)
$\|D\| \leftarrow 0$ {Initialize values of the distance matrix, $D$.}
for $i \leftarrow 1$ to # of rows in $\|T\|$ do
    for $k \leftarrow 1$ to # of rows in $\|QI\|$ do
        for $j \leftarrow 1$ to # of attributes in $\|QI\|$ do
            if $T^j_j = QI^i_j$ then
                $D^i_k \leftarrow D^i_k + 1$
            end if
        end for
        $D^i_k \leftarrow \frac{D^i_k}{\#ofattributesinQI}$
    end for
end for
{Compute privacy metrics $PM_1$ and $PM_2$.}
$R \leftarrow 0$
$diffRecords \leftarrow 0$
$d \times d \leftarrow \|D\|$ {Dimensions of the similarity matrix.}
for $i \leftarrow 1$ to $d$ do
    if $D(i,i) < 1$ then
        $diffRecords \leftarrow diffRecords + 1$ {For metric $PM_2$.}
    end if
    for $j \leftarrow 1$ to $d$ do
        if $i \neq j \wedge D(i,j) \geq D(i,i)$ then
            $R \leftarrow R + 1$
        end if
    end for
end for
return $PM_1 = \frac{R}{d}$, $PM_2 = \frac{diffRecords}{d}$
```

An example of application of the algorithm `PriestPrivacy` is shown in Table 3 that contains (for illustrative purposes) three attributes: `Age`, `Gender`, and `Race`. The first column of this table holds the record number. Original values are shown in each cell with their sanitized replacements shown as a superscript for each value. For example, record `1` contains original value `30` for the attribute `Age` that is anonymized and replaced with the value `40`, which is one of three distinct values for this attribute. The superscripted values populate the matrix $\|T\|$. Interestingly, the sanitized record `1` matches the original record `2`; however, since the attacker sees only the sanitized data, it is not possible for the attacker to know with certainty that the sanitized record matches some original record.

4.4 Privacy Metrics

We first calculate the probability distribution of guessing anonymity over all the database records. Specific privacy metrics are then defined as a function of that distribution. In this paper, we define three of them but the optimal metric may vary based on the task at hand.

A privacy metric measures how identifiable records in the sanitized table are with respect to the original table [21, page 43]. The privacy metrics that we propose in this paper are all motivated by the notion of guessing anonymity. We calculate guessing anonymity using the algorithm shown in Algorithm 2. We begin by creating a similarity matrix that shows the similarity between sanitized and original records. For each record R_o in the original table, the similarity of R_o to a record R_s in the sanitized table is

Table 3: A table with original and anonymized (superscript) data. For example, the original value of the attribute Age is 30 and the sanitized value is 40.

Record	Age	Gender	Race
Rec 1	30^{40}	F^{M}	W^{B}
Rec 2	40^{40}	M^{M}	B^{H}
Rec 3	45^{30}	M^{F}	H^{W}
Rec 4	30^{40}	F^{M}	W^{H}

Table 4: A similarity matrix for the table shown in Table 3. Each cell contains the value that shows the fraction of attributes values that are the same between original and sanitized records.

Original Anonymized	Rec 1	Rec 2	Rec 3	Rec 4
Rec 1	0	1	0.33	0
Rec 2	0	0.66	0.66	0
Rec 3	1	0	0	1
Rec 4	0	0.66	0.66	0

measured by the fraction of attributes whose values are the same between R_o and R_s. This computation is performed in Lines `3-12` of the privacy metric algorithm. This similarity matrix, $||D||$, shows similarities between rows in the original matrix, $||QI||$, and the anonymized matrix, $||T||$.

Table 4 shows an illustrative example for computing the similarity matrix $||D||$ for the data in Table 3. The similarity matrix $||D||$ has dimensions $d \times d$, where d is the number of records in the matrices $||QI||$ and $||T||$. Rows correspond to anonymized records and columns correspond to the original records in $||D||$. The values of the attributes `Age`, `Gender`, and `Race` are `30`, `F`, and `W` for the original record `1`, respectively, and the values for the sanitized record `1` are `40`, `M`, and `H`, respectively. Therefore, the similarity score is zero for the original and sanitized record `1`. In contrast, the similarity score is one for original record `2` and sanitized record `1` since all attribute values of the original record 2 match the values of their corresponding attributes for the sanitized record `1`.

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 $||D||$ is a diagonal matrix, i.e., all entries outside the main diagonal are zero. In this case, each record is similar to itself only, that is, all records are unique and easily identifiable by attackers. The privacy level for this sanitized table is zero. The other extreme case is when all sanitized records are similar to all other records. In this case, it is very difficult for attackers to guess original data since all sanitized records are highly similar to one another. Correspondingly, the privacy level for this sanitized table is close to one. In practice, the privacy level is between zero and one, and our goal is to help analysts find the right balance between a privacy level and the utility of the sanitized database.

The distribution of guessing anonymity of the different records in a database can be used to define a variety of privacy metrics. In this paper, we propose three different privacy metrics that are derived from the guessing anonymity distribution, but do not focus on providing the *best* metric. We believe that different applications and risk tolerances of users would require the use of different derived metrics and leave the optimal metric determination as future work.

In this paper, we propose three metrics based on guessing anonymity: PM_1: *Mean guessing anonymity*, PM_2: *Fraction of records with guessing anonymity greater than a certain value m (where we set m to one in this paper)*, and *Unique Records*. The computation of these privacy metrics PM_1 and PM_2 is shown in Lines `14-27` of Algorithm 2, and their values are returned in Line `27`.

PM_1: The *Mean guessing anonymity* of a database is the arithmetic mean of the individual guessing anonymities for each record in the database. This metric gives us measurement of the overall privacy of the sanitized database and helps us compare different versions of sanitized databases. Naturally, a database with higher *Mean guessing anonymity* would be more difficult to attack and hence have higher privacy. Suppose that there is a record $r_o \in T_o$, where T_o is the table with original records. After T_o is sanitized, the table T_s with sanitized records is obtained, $r_s \in T_s$, and we compute the similarity matrix, $||D||$, for records in these tables. Then, for each $r_o \in T_o$, we increase the counter, R, by one if we find a record in the table T_s that has the similarity score in $||D||$ equal to or higher than the similarity score between r_o and its sanitized version, r_s, not counting the similarity score between r_o and r_s. The formula for the privacy metric is $PM_1 = \frac{R}{d}$, where d is the number of records.

PM_2: The fraction of records with guessing anonymity greater than m is our second metric. We set m to one for the work in this paper; such setting corresponds to measuring the fraction of original records that have been modified at all by the anonymization algorithm. PM_2 is calculated as the ratio of records that were sanitized and that differ from their original record in at least one attribute value to the total number of records. This metric is useful for a variety of reasons. If the fraction of records that have guessing anonymity greater than m is too low and the users are unsatisfied, they have the option to increase the level of privacy or remove the records that fall below the threshold from the sanitized database, thus making the database more private.

UniqueRecords is measured as part of PM_2 and it is defined as follows. Suppose that there is a record $r_o \in T_o$, where T_o is the table with original records. After T_o is sanitized, the table T_s with sanitized records is obtained. Let $r_s \in T_s$ be the sanitized record for $r_o \in T_o$. r_s is a unique record if there exists $r'_o \in T_o$, such that all attributes of r'_o have the same value as the attributes of r_s. A key idea of guessing anonymity is that although some records after applying data swapping may match some original records, the attacker still cannot know with certainty which ones do and whether sensitive information (that these records identify) was not changed. Ideally, the percentage of unique records in the anonymized database will be zero - that is what we would like to achieve. If the number of unique records is greater than zero and the desired goal is to have complete anonymity, the users will have to delete these unique records before releasing the database.

5. EXPERIMENTAL EVALUATION

In this section, we describe the results of the experimental evaluation of PRIEST on three open-source Java programs and one large commercial application that is used to manage logistics of one of the largest supermarket chains in Spain.

5.1 Research Questions

In this paper, we make one meta-claim – our privacy framework for managing the tradeoff between data privacy and test coverage for DCAs is "better" than other frameworks. We define "better" in two ways: *coverage* and *flexibility*. Achieving higher coverage with PRIEST means that for a given level of privacy, we can provide higher test coverage. We measure it using the area under the privacy-coverage curve. Flexibility means that we can give more choices to the business analyst to make informed decisions. If more data points in that curve can be created using PRIEST for a given range of privacy, we have a better framework for managing the tradeoff.

PRIEST does not compete with other anonymization techniques, such as *k*-anonymity since the latter is a metric, not a framework. Our claim is that our framework (that includes guessing anonymity) is a better framework to achieve coverage and flexibility. We seek to answer the following research questions.

RQ1 How much test coverage does PRIEST help achieve at given levels of privacy for subject applications?

RQ2 How effective is PRIEST in achieving different levels of data privacy while preserving original data?

RQ3 How effective is PRIEST in releasing different versions of anonymized databases for the same level of privacy and test coverage without enabling the attacker to link sensitive data?

With RQ1, we address our claim that we designed and implemented a technique using program analysis for determining how values of database attributes affect test coverage of DCAs that use this data. Our goal is to show that with PRIEST, the privacy metric is linked directly to test coverage and vice versa; in other words, guaranteeing a certain level of test coverage should allow stakeholders to calculate bounds of the privacy level.

With RQ2, we address our claim that using different levels of privacy enables business analysts to make trade-off decisions about privacy and utility.

With RQ3, we address our claim that PRIEST enables stakeholders to repeatedly release anonymized data in such a way that both privacy and utility are guaranteed at certain levels. Suppose that a technique of data suppression anonymization is applied to protect data and release the anonymized data with the DCA to testers. After some time, programmers produce the next release of this DCA that uses different attributes differently in its database. Suppose that the DCA does not use the attribute `Nationality` any more, and it uses the attribute `Race` instead. At this point, the database should be reanonymized, since the original values of the attribute `Race` should be left intact to preserve test coverage. However, in the previous anonymization, the values of the attribute `Nationality` are left unprotected. It means that the testers (the attackers) know original values of the attribute `Nationality` and now the testers will know the original values of the attribute `Race`. Together, these attributes enable the attackers to infer sensitive information from the released database.

5.2 Subject Programs

We evaluate PRIEST with three open-source and one large commercial Java programs that belong to different domains. 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.

We selected four subject programs that come with test cases. `N2A` is a program for handling logistics of one of the largest supermarket chains in Spain. `N2A` has a total of 77,828 LOC. `RiskIt` is an insurance quote program[4]. `DurboDax` enables customer support centers to manage customer data[5]. Finally, `UnixUsage` is a program for obtaining statistics on how users interact with Unix systems using their commands[6].

DCA	App	Test	DB	Tbl	Att	IC	ETC
	[kLOC]		[MB]			%	%
N2A	77.8	19.3	20	506	2514	53	50
DurboDax	2.8	2.0	49	27	114	77	59
UnixUsage	2.8	.9	21	8	31	61	53
RiskIt	4.3	2.6	628	13	57	62	48

Table 5: Characteristics of the subject DCAs. App = application code, Test = test cases, DB = database, Tbl = tables, Att = attributes in all tables, IC = initial test coverage with the original database, ETC = estimated worst test coverage with sanitized data using the approach described in Section 3.

Table 5 contains characteristics of the subject programs, 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 project ranges from 2.8 to 77.8 kLOC. The test cases range from 0.9 to 19.3 kLOC. Other columns show the size of the database, number of tables and attributes in the database, test coverage (statement coverage) that is obtained with the original database, and the estimated worst test coverage (statement coverage) with sanitized data using our approach described in Section 3.

5.3 Methodology

To evaluate PRIEST, we carry out experiments to explore its effectiveness in enabling users to determine how to balance test coverage while achieving different levels of data privacy (RQ1 and RQ2), and to show that it is possible to apply PRIEST to get privacy guarantees while releasing different sanitized versions of the database for the same privacy levels thereby supporting software evolution (RQ3).

5.3.1 Variables

The main independent variable is the value of p. Two main response variables are the time that it takes to anonymize data in the database to achieve the desired level of k to answer RQ2 and the test coverage in percentage of program statements to answer RQ1.

5.3.2 The Structure of the Experiments

For the experiments, we select as QIs all attributes whose values affect execution paths in the corresponding DCAs. It is physically not possible to carry out an experiment using all subsets of the powerset of attributes as QIs where we measure test coverage for subject DCAs while achieving anonymity, since it would require us to consider the powerset of all attributes. Given that databases of the subject DCAs contain between 31 and 2,514 attributes, it is challenging to select a subset of them as QIs to enable achieve the higher possible level of test coverage.

Our goal is to run experiments for different values of the independent variable p and report the effect of varying the values on p on dependent variables. To address RQ3, we anonymize databases for the subject DCAs for a given level of p, and we report and analyze privacy metrics between different sanitized versions of the same original database.

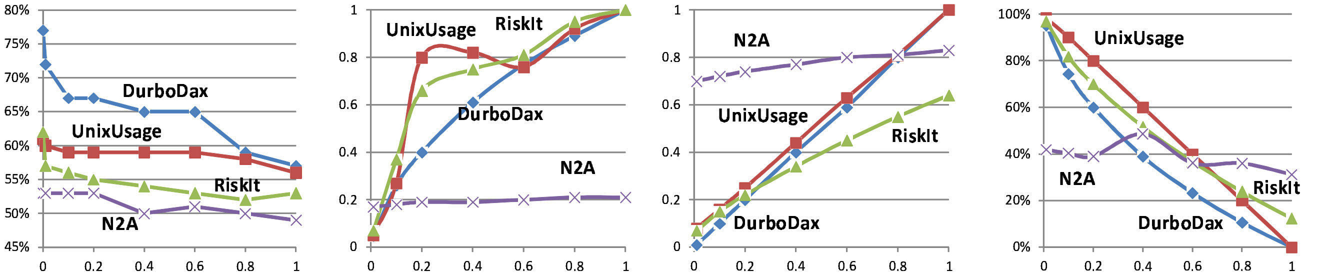
5.4 Threats to Validity

A threat to the validity of this experimental evaluation is that our subject programs are of small to moderate size because it is difficult to find a large number of open-source programs that use nontrivial databases. Large DCAs that have millions of lines of code and use databases whose sizes are measured in thousands of

[4] https://riskitinsurance.svn.sourceforge.net as of March 10, 2011.

[5] http://se547-durbodax.svn.sourceforge.net as of March 10, 2011.

[6] http://sourceforge.net/projects/se549unixusage as of March 10, 2011.



(a) Test coverage.

(b) Privacy metric PM_1.

(c) Privacy metric PM_2.

(d) Unique records.

Figure 3: Experimental results for the subject DCAs. All graphs show the dependence on the probability of replacing original data value that is assigned to the horizontal axis. The vertical axis designates dependent variables that are given in the captions to these figures.

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

5.5 Results

The results of the experiments conducted to address $RQ1$ and $RQ2$ are shown in Figure 3 and Table 6. Dependency of test coverage on the probability p of replacing original values is shown in Figure 3(a). The maximum reduction in test coverage is close to 26% from the initial test coverage with $p = 1$ for application DurboDax. In our previous work [23], we showed that the maximum reduction in test coverage for the same applications reaches 80% from the initial test coverage when using a popular algorithm Datafly that is based on data suppression and generalization techniques [36]. We observed the biggest drop in test coverage with Datafly when $k = 7$, while much smaller maximum drop is observed with PRIEST for $p = 1$, which is the maximum value for the anonymization parameter.

The least drop in test coverage is observed for the application N2A, while the biggest drop is attributed to the application DurboDax. Our explanation is that N2A is least sensitive to values of the database attributes since it uses these values to compute results rather than in path conditions. We explain the sharper drop for DurboDax as a result of branch conditions that use conjunctions of multiple QIs, thereby making the application sensitive to joint distributions of values of different attributes that may be destroyed by anonymization. We also observe that the shape of the test coverage curves show gradual decline in test coverage rather than abrupt changes. The former makes it easier for stakeholders to balance privacy and coverage on a continuous scale.

Dependencies of privacy metrics PM_1 and PM_2 on the probability p of replacing original values are shown in Figure 3(b) and Figure 3(c). Combined with the test coverage curve shown in Figure 3(a), these dependencies enable business analysts to select a level of protection that also ensures a certain level of test coverage. While values of PM_1 are monotonically increasing for N2A, RiskIt, and DurboDax with the increase in p, the values of PM_1 for UnixUsage slightly drop when the values of p are increasing from 0.2 to 0.6. This result can be explained as an effect of the variations of random value replacement that may result in small changes in the numbers of similar records. The values for $PM1$ are relatively small for N2A since many of the selected QIs were involved in some primary or unique key. As a result, the sanitized records contain distinct values for these QIs and fewer records (in the original database) are likely to be similar to a sanitized record.

Table 7: Results for privacy metrics to compare two sanitized databases for $p = 0.6$ for subject DCAs

Subject DCA	PM_1	PM_2	Total Records	Unique Records	$UR,\%$
N2A	0.20	0.78	586801	265400	38.7
DurboDax	0.60	0.77	88450	20687	23.4
UnixUsage	0.61	0.60	250570	100422	40.1
RiskIt	0.47	0.76	1270792	512051	40.3

Opposite to the dependencies of privacy metrics PM_1 and PM_2, the graph of the dependency of the percentage of unique records, UR shows corresponding decline in Figure 3(d). It means that the percentage of unique records between the original and sanitized databases is monotonically declining as the values of the probability p are increasing thereby making it more difficult for attackers to use these unique records to guess the original data. While values of UR are monotonically decreasing for UnixUsage, RiskIt, and DurboDax with the increase in p, the values of UR for N2A slightly increase when the values of p are increasing from 0.2 to 0.6. This result can be explained as an effect of the variations of random value replacement that may result in small changes in the numbers of unique records.

To address RQ3, we generated two sanitized databases for $p = 0.6$ and computed the privacy metrics $PM1$, $PM2$, and number of unique records between the two databases. These metrics quantify the difficulty of an attacker to relate records from the two sanitized databases. Since the same database is anonymized independently and the probabilities of replacing cell values are independent from one another, then resulting databases cannot be deterministically correlated since they are not used to produce each other. Thus, the attacker has to guess original data independently, meaning that the lowest guessing anonymity score can be used to quantify the difficulty of an attacker to guess records from both sanitized databases.

The values for $PM1$ are slightly lower to the values of $PM1$ for $p = 0.6$ in Table 6, while the values of $PM2$ are slightly higher to the values of $PM1$ for $p = 0.6$ in Table 6. This phenomenon is expected since the variation between the records of the two anonymized databases is expected to be more as compared to variation between the records of original and an anonymized database. The number of unique records is similar to the number of unique records for $p = 0.6$ in Table 6. It is also important to note that the overall privacy of multiple sanitized databases is only as good as the privacy of the least private version. If the first released ver-

Subject	QIs	Total	Initial	Worst	Max	Dependent	The Probability of Value Replacement, p						
		Records	Cov	Exp Cov	Err Cov	Variable	0.01	0.1	0.2	0.4	0.6	0.8	1
N2A	61	586801	53%	50%	2.0%	PM_1	0.17	0.18	0.19	0.19	0.20	0.21	0.21
						PM_2	0.70	0.72	0.74	0.77	0.80	0.81	0.83
						TC,%	49.00	49.00	49.00	50.00	51.00	49.00	49.00
						UR,%	41.80	40.30	39.00	48.60	36.10	36.00	31.30
DurboDax	20	88450	77%	59%	3.6%	PM_1	0.05	0.26	0.40	0.61	0.77	0.89	1.00
						PM_2	0.01	0.10	0.20	0.40	0.59	0.80	1.00
						TC,%	72.00	67.00	67.00	65.00	65.00	59.00	57.00
						UR,%	95.20	74.40	60.00	38.90	23.30	10.60	0.00
UnixUsage	16	250570	61%	53%	5.7%	PM_1	0.05	0.27	0.80	0.82	0.76	0.92	1.00
						PM_2	0.08	0.16	0.25	0.44	0.63	0.81	1.00
						TC,%	60.00	59.00	59.00	59.00	59.00	58.00	56.00
						UR,%	98.30	90.00	80.00	60.20	39.90	20.00	0.01
RiskIt	28	1270792	62%	48%	8.3%	PM_1	0.07	0.37	0.66	0.75	0.81	0.95	1.00
						PM_2	0.07	0.15	0.22	0.34	0.45	0.55	0.64
						TC,%	57.00	56.00	55.00	54.00	53.00	52.00	53.00
						UR,%	96.90	81.70	69.90	51.60	36.70	23.80	12.30

Table 6: Results of experiments with subject DCAs for different values of the independent variable p that is shown in the last column header that spans seven subcolumns. The second column, QI, shows the number of QI whose values affect control-flow decisions in DCAs, the next two columns show the initial test coverage (statement coverage) with the original database and the estimated worst test coverage with sanitized data using the approach described in Section 3. The next column shows the error in the estimated test coverage that varies from 2.0% to 8.3%. Finally, the next column lists four dependent variables, privacy metrics PM_1 and PM_2, test coverage, TC, and the percentage of unique records, UR. Finally, the last seven subcolumns show values of these dependent variables for different values of p.

sion has PM_2 = 3 and the second version has PM_2 = 5, PM_2 for the two databases combined will be 3. In general, for two released databases D_i and D_j, the overall PM_2 = Min($PM_2(D_i)$, $PM_2(D_j)$).

Result summary. These results strongly suggest that PRIEST helps achieve higher test coverage for given levels of privacy for subject applications when compared with Datafly that is based on data suppression and generalization techniques, thereby addressing RQ1. PRIEST can also achieve different levels of data privacy while preserving original data as it is seen from Table 6, thereby addressing RQ2. Finally, the results shown in Table 7 strongly suggest that PRIEST is effective in releasing different versions of anonymized databases for the same level of privacy and test coverage, thereby addressing RQ3.

6. RELATED WORK

Our work is related to regression testing [39] since PRIEST 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.

Closely related to PRIEST 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 [10]. Unlike PRIEST, $kb-$anonymity replaces some information in the original data to ensure privacy preservation so that the replaced data can be released to third-party developers. PRIEST and $kb-$anonymity are complementary in using different privacy mechanisms to preserve original data thereby improving its testing utility.

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, PRIEST 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 k-anonymity do not allow the user to explicitly control how much each record is altered. Empirical results reported by Rachlin et al. [31] 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 [2]. 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.

7. CONCLUSION

We offer a novel and effective approach called PRIEST that helps organizations to remove an obstacle to effective DCA test outsourcing. With PRIEST, DCAs can be released to external testing organizations without disclosing sensitive information while retaining testing efficacy. We built a tool and applied it to nontrivial DCAs. The results show that PRIEST is effective in enabling users to determine how to balance test coverage while achieving different levels of data privacy, and that with PRIEST, users can release different sanitized versions of the database for the same privacy levels, thereby supporting software evolution.

Acknowledgments

We thank anonymous reviewers whose comments helped us to improve the quality of this paper. This work is supported in part by NSF CCF-0916139, CCF-1017633, CCF-0725190, CCF-0845272, CCF-0915400, CNS-0958235, an NCSU CACC grant, and ARO grant W911NF-08-1-0443, and Accenture. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

8. REFERENCES

[1] C. C. Aggarwal. On k-anonymity and the curse of dimensionality. In *VLDB*, pages 901–909, 2005.

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

[3] C. C. Aggarwal and P. S. Yu. On static and dynamic methods for condensation-based privacy-preserving data mining. *ACM Trans. Database Syst.*, 33:2:1–2:39, March 2008.

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

[5] J. T. Alexander, M. Davern, and B. Stevenson. Inaccurate age and sex data in the census pums files: Evidence and implications. Working Paper 15703, National Bureau of Economic Research, January 2010.

[6] S. Ambler. Robust persistence layers. *Softw. Dev.*, 6(2):73–75, 1998.

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

[8] C. Bialik. Census bureau obscured personal data – too well, some say. *The Wall Street Journal*, Feb. 2010.

[9] J. Brickell and V. Shmatikov. The cost of privacy: destruction of data-mining utility in anonymized data publishing. In *KDD*, pages 70–78, 2008.

[10] 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.

[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] G. Cormode and D. Srivastava. Anonymized data: generation, models, usage. In *SIGMOD*, pages 1015–1018, 2009.

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

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

[18] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, pages 202–210, 2003.

[19] J. Domingo-Ferrer and D. Rebollo-Monedero. Measuring risk and utility of anonymized data using information theory. In *EDBT/ICDT*, pages 126–130, 2009.

[20] A. Endres and D. Rombach. *A Handbook of Software and Systems Engineering*. Pearson Addison-Wesley, 2003.

[21] 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.

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

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

[24] M. Grechanik, C. McMillan, L. DeFerrari, M. Comi, S. Crespi, D. Poshyvanyk, C. Fu, Q. Xie, and C. Ghezzi. An empirical investigation into a large-scale Java open source code repository. In *ESEM*, pages 11:1–11:10, 2010.

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

[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] W. Landi. Undecidability of static analysis. *ACM Lett. Program. Lang. Syst.*, 1(4):323–337, 1992.

[28] T. Li and N. Li. On the tradeoff between privacy and utility in data publishing. In *KDD*, pages 517–526, 2009.

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

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

[31] Y. Rachlin, K. Probst, and R. Ghani. Maximizing privacy under data distortion constraints in noise perturbation methods. In *PinKDD*, pages 92–110, 2008.

[32] S. P. Reiss. Practical data-swapping: the first steps. *ACM Trans. Database Syst.*, 9:20–37, March 1984.

[33] S. P. Reiss, M. J. Post, and T. Dalenius. Non-reversible privacy transformations. In *PODS*, pages 139–146, 1982.

[34] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. Knowl. Data Eng.*, 13(6):1010–1027, 2001.

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

[36] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

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

[38] B. G. Thompson. *H.R.6423: Homeland Security Cyber and Physical Infrastructure Protection Act of 2010*. U.S.House, 111th Congress, Nov. 2010.

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