

## Is Data Privacy Always Good For Software Testing?

Mark Grechanik  
*University of Illinois at Chicago*  
 Chicago, IL 60601  
 drmark@uic.edu

Christoph Csallner  
*University of Texas at Arlington*  
 Arlington, TX 76019-0015  
 csallner@uta.edu

Chen Fu and Qing Xie  
*Accenture Technology Labs*  
 Chicago, IL 60601  
 {chen.fu, qing.xie}@accenture.com

**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 releasing proprietary DCAs, its databases should also be made available to test engineers, so that they can test using real data. Testing with real data is important, since fake data lacks many of the intricate semantic connections among the original data elements. However, different data privacy laws prevent organizations from sharing these data with test centers because databases contain sensitive information. Currently, testing is performed with fake data that often leads to worse code coverage and fewer uncovered bugs, thereby reducing the quality of DCAs and obliterating benefits of test outsourcing.*

We show that a popular data anonymization algorithm called  $k$ -anonymity seriously degrades test coverage of DCAs. We propose an approach that uses program analysis to guide selective application of  $k$ -anonymity. This approach helps protect sensitive data in databases while retaining testing efficacy. Our results show that for small values of  $k \leq 6$  it is possible to retain a higher test coverage with our approach than with the current state-of-the-art  $k$ -anonymization algorithm, Datafly. However, with either our approach or Datafly, for  $k \geq 7$ , test coverage drops to less than 30% from the original coverage of more than 70%, thus making it difficult to achieve good quality when testing DCAs while applying data privacy.

### I. INTRODUCTION

*Database-centric applications (DCAs)* are common in enterprise computing, and they use nontrivial databases [1]. Some DCA owners are large organizations such as banks, insurance companies and government agencies. Their databases usually contain private data of many individuals. If a large organization needs to test a new custom software application, it typically hires a software consulting company to provide testing services. When releasing proprietary DCAs to test centers, it is also desirable that databases are made available, so that testing can be conducted using real data. Over the years, some DCA owners have developed special relations with “their” test centers, which often implies a higher level of trust. This trust enables the transfer of sensitive data from DCA owners to test centers. However, it becomes harder to rely on such trust relationships when faced with recent data protection laws and regulations around the world [2]. Currently, data owners can no longer easily share confidential data with external software service providers. In this paper we address the

problem of how to protect sensitive data while preserving test quality.

Real data is very important for high quality testing. Original data elements are often connected via intricate semantic relationships that are not always explicitly defined in database schema. 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 [3]. In many cases, domain experts review results of testing manually, since generated data often leaves the expected results unspecified [4, pages 114, 116]. In many cases data generators miss important relationships between data elements, and running applications with test cases that use this synthetic data produces results that make little sense to these domain experts. For example, what would be a good oracle for medical insurance software if generated input test data describes a male who suffers from a form of gestational diabetes that can occur only during pregnancy? Computing insurance premium makes no sense using this data. In reality, there are multiple relationships among data elements, many of which are far from obvious. Thus it is important to release real data to testers while protecting sensitive information.

Because of recently tightened data protection regulations, test centers no longer get access to sensitive data. Test engineers have to operate with little or no meaningful data, it is an obstacle to creating test suites that lead to good quality software. As a result, test centers report worse code coverage and fewer uncovered bugs, thereby reducing the quality of applications and obliterating benefits of test outsourcing [5].

As an example, consider a situation where an insurance company outsources testing of its newly developed medical claim application to an external test center. The insurance company accumulated historical data on its users, and it is highly desirable that this data is used when testing the claim application. For example, selecting individuals whose nationalities are correlated with higher rates of certain diseases yield extensive computations, and creating test cases that involve data on these individuals may lead to more path coverage and they are likely to reveal more bugs [6], [7]. However, when a company sanitizes (or anonymizes) this data, test coverage is likely to worsen. For instance, replacing the values of nationalities with the generic value

“Human” may lead DCAs to execute some default paths that result in exceptions or miss certain paths resulting in worse test coverage. From this example we can see that applying data privacy is not generally good for software testing. Recent cases with U.S. Census show that applying data privacy leads to incorrect results [8], [9]. Therefore preserving test coverage while achieving desired data anonymity is not an easy task.

Given the importance of this problem, it may be surprising that there is little prior research on this topic. There may be two main reasons for this. First, elaborate data privacy laws are a new phenomenon, and many of these laws have been introduced after the year 2000. Second, it is only in the past decade that applications are increasingly being tested by 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 [10], [11].

#### A. State of the Art

After interviewing professionals at IBM, Accenture, two large health insurance companies, a biopharmaceutical company, and three major banks we found that current test data anonymization processes are manual, laborious, and error-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. Test engineers from outsourcing companies come to the cleanroom of the client company to test client’s DCAs. Actions of these test engineers are tightly monitored; electronic connections to outside of the company’s network, phone calls, USB keys, and cameras 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. This procedure 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. By repopulating large databases with fake data it is likely that many implicit dependencies and patterns among data elements are omitted, thereby reducing testing efficacy. Moreover, fake data are likely to trigger exceptions in DCAs leading test engineers to flood bug tracking systems with false error 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.

A more sophisticated approach is *selective anonymization*, where a team is assembled that comprises business analysts and database experts. After they set privacy goals, identify sensitive data, and mark database attributes that may help attackers to reveal this sensitive data (these attributes are called *quasi-identifiers (QIs)*), anonymization techniques are applied to these QIs to protect sensitive data, resulting in a sanitized database. For example, consider a health information database that holds information about medical history of individuals. In this case, the attribute that holds the names of diseases is considered to have sensitive information. Other attributes that hold information about individuals (e.g., age, race, nationality) are viewed as QIs. Knowing values of some of QIs enables attackers to deduce sensitive information about individuals who are identified by these values. A goal of all anonymization approaches is to make it impossible to deduce certain facts about entities with high confidence from the anonymized data [12, pages 137-156].

#### B. Problems and Goals

A fundamental problem of test outsourcing is how a DCA owner can release its private data with guarantees that the subjects of the data (e.g., people, equipment, policies) cannot be re-identified using their attributes while the data retain their testing efficacy. Ideally, sanitized data should induce execution paths that are similar to the ones that are induced by the original data. That is, when sanitizing data, information about how DCAs use this data should be taken into account.

In many cases, in order to protect sensitive information not all of the identified QIs need to be anonymized. For example, consider a situation where an individual can be identified by four QIs: Age, Race, Nationality, and ZipCode. However, depending on privacy goals, only one of two combinations of Age and Race or Nationality and ZipCode need to be anonymized to protect the identity of an individual. At the same time, if it is known that a DCA uses the values of the attribute Age, anonymizing the values of Nationality and ZipCode is likely to have no effect on executing this DCA. Therefore it is important to know how DCAs use their databases to which these algorithms are applied in order to decide which attributes to select as QIs.

Finally, different DCAs have different privacy goals and levels of data sensitivity. Applying more relaxed protection to databases is likely to result in greater test coverage since a small part of the database will be anonymized; conversely, stricter protection makes it more difficult to outsource testing. The latter is the result of two conflicting goals: make testing as realistic as possible and hide real data from testers who need this data to make testing effective. A problem is how to balance these goals, i.e., to anonymize values of database attributes while preserving test coverage of DCAs that use these values.

Rec	Age	ZipCode	Nationality	Disease
1	42	52000	American	Ulcer
2	47	53000	Palauan	Viral
3	51	32000	American	Heart disease
4	55	32000	Japanese	Gastritis
5	62	51000	Palauan	Dyspepsia
6	67	35000	American	Dyspepsia

↓ **k-anonymity**

Rec	Age	ZipCode	Nationality	Disease
1	50	50000	Human	Ulcer
2	50	50000	Human	Viral
3	50	30000	Human	Heart disease
4	20	30000	Human	Gastritis
5	50	50000	Human	Dyspepsia
6	20	30000	Human	Dyspepsia

Figure 1. Example of applying  $k$ -anonymity to the top table with  $k = 2$  that results in a sanitized lower table.

For *selective anonymization*, after privacy goal is set, selecting which QIs to anonymize is the only degree of freedom when applying anonymization algorithms. Thus, it also becomes the only point of attack if we want to preserve test coverage without compromising privacy goal.

### C. Our Contributions

This paper makes the following contributions:

- We offer a novel approach, *Testing Applications with Data Anonymization (TaDa)* with which organizations can determine how much test coverage they can lose when applying data privacy to DCAs.
- We experiment with four nontrivial DCAs and we selected the most popular data privacy approach called  $k$ -anonymity, where each entity in the database must be indistinguishable from  $k - 1$  others [13]. We show that for small values of  $k \leq 6$  (i.e., lower levels of data privacy) it is possible to achieve higher test coverage with our approach than with the current state-of-the-art  $k$ -anonymization algorithm, Datafly. However, we show that for higher values of  $k \geq 7$  that lead to much stricter data privacy levels test coverage drops to less than 30% from the original coverage of more than 70%, thus making it difficult to achieve good quality when testing DCAs while applying data privacy.

## II. THE PROBLEM

In this section we provide the necessary background on DCAs and data anonymization, show how sanitizing data affects test coverage of DCAs, and formulate the problem statement.

### A. Background

Majority of enterprise-level DCAs use 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 allows DCAs to access database engines through standardized application programming interfaces (APIs) (e.g., Java DataBase Connectivity (JDBC)) [14]. Using JDBC, SQL queries are passed as strings to corresponding API calls to be sent to databases for execution.

Once these queries are executed, the values of attributes of database tables are returned to DCAs, which in turn use these values as part of their application logic. This means, values from the database may be used in branch decisions, loop conditions, etc. and they 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. Removing certain classes of values in database attributes may make some branches and statements in DCAs unreachable.

At the same time certain classes of values of database attributes that contain non-sensitive information should be anonymized. These attributes (i.e., 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 approaches [15]. 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 values of attributes of tables, and a common side-effect of these modifications is unreachable statements in DCAs that are otherwise executed with the original data.

### B. A Motivating Example

Consider a motivating example that shows how applying anonymization approaches to data makes operations unreachable in DCAs. The original database table is shown in the upper part of Figure 1. It contains three QIs: *Age*, *ZipCode*, and *Nationality*. The attribute *Disease* contains sensitive data about diseases of individuals. Even though the names of individuals are not given in this table, it is possible to identify them using the values of the QIs and link these individuals to specific diseases.

This table exposes an individual whose data is shown in row five. The values of the QIs specify that the individual

```

if (nationality=="Palauan" && age > 60) {
    f(disease);
}

```

Figure 2. A motivating example that shows how applying anonymization approaches to the data as shown in Figure 1 makes the function  $f$  unreachable.

is a 62-year old Palauan who lives in zip code 51000. If we know that there is a single 62-year old Palauan who lives in this zip code, we can infer that this person suffers from dyspepsia.

To protect this information, we  $k$ -anonymize this table for the lowest possible value of  $k = 2$  by suppressing the values of the QI `Nationality`, perturbing the values of the QIs `Age`, and generalizing the values of `ZipCode`. The resulting table is shown in the lower part of Figure 1. For each row in this table there is at least one other row that contains the same values for the QIs, making it difficult to infer with certainty what individual has which specific disease. However, statements that used to be reachable with the original data may not be reachable with the anonymized data.

Consider the fragment of code shown in Figure 2. After executing JDBC API calls, the values of the QIs `Age`, `Nationality`, and `Disease` are put in the corresponding variables of the DCA `nationality`, `age`, and `disease`. Since the values of these QIs are modified, the function call  $f()$  becomes unreachable. Clearly, applying data privacy is not good for testing this example.

### C. The Problem Statement

The problem statement that we address in this paper is how to preserve test coverage for DCAs while achieving desired privacy goals for databases that these DCAs use for  $k$ -anonymity. In this paper we concentrate on the statement coverage criterion that states that all statements are covered if and only if for all nodes  $n \in N$  in the control-flow graph of some DCA, there is at least one path  $p \in P$  such that that node  $n$  is on the path  $p$ , where  $P$  is the set of all paths executed during testing the DCA and  $N$  is the set of all nodes in the control-flow graph [7].

Suppose that  $N_o \subseteq N$  is the set of all nodes that are covered when testing a DCA with the original data, and  $N_a \subseteq N$  is the set of all nodes that are covered when testing the same DCA with the anonymized data. In general, testing with  $N_a$  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 bypassed branches. For example, the DCA logic handles suppressed values of `Nationality` by bypassing the body of the `if` statement as it is shown in Figure 2. 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.

Ideally, all statements (i.e., nodes in the call graph) that are executed with original data should also be executed with anonymized data. What makes it difficult to reach our goal is that it is an undecidable problem to determine precisely how values of QIs are used in DCAs [16]. Without knowing traces between QIs and program variables it is difficult to compute the effect of replacing values of these QIs on test coverage of the program.

Our goal is to investigate how to select a set of QIs as a subset of attributes of tables in a database. Different data anonymization approaches use different heuristics on how to select attributes as quasi-identifiers [17]. For example, a popular heuristics for the Datafly algorithm for  $k$ -anonymity is to select attributes that have a large number of distinct values, while the algorithm Mondrian advocates selection of attributes with the biggest range of values [18]. By selecting different QIs for applying  $k$ -anonymity it is possible to change test coverage. It is important to know how DCAs use values of attributes, since anonymizing values of attributes that are not used by DCAs is unlikely to affect test coverage of these DCAs. However, there is no heuristics that considers how attributes are used by DCAs to preserve test coverage.

## III. OUR SOLUTION

In this section, we present core ideas behind our approach that we call *Testing Applications with Data Anonymization (TaDa)* and we describe the TaDa architecture.

### A. Core Ideas

Our core idea to link attributes of the database with the DCA that uses this database. To address this idea we use a technique that tracks how different database values affect the application’s behavior. That is, TaDa links program variables automatically to database attributes. The values of these attributes can be used in conditional expressions to make branching decisions, thereby influencing the execution flow of the DCA. Our goal is to quantify the effect of replacing values of database attributes on reachability of program statements.

Using our idea unifies applications and their databases in a novel way: database attributes are tied to the source code of the DCAs and how these DCAs use values of these attributes determines 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. Specifically, the value of  $k$  in  $k$ -anonymity designates that  $k$  entities are indistinguishable from one another. The lowest value of  $k = 2$  offers the least protection, and higher values of  $k$  offer more protection. For example, protecting the database of movie ticket buyers may require much lower protection than the database that holds medical information. Therefore it is important to extend anonymization algorithms that implement data protection with information about how

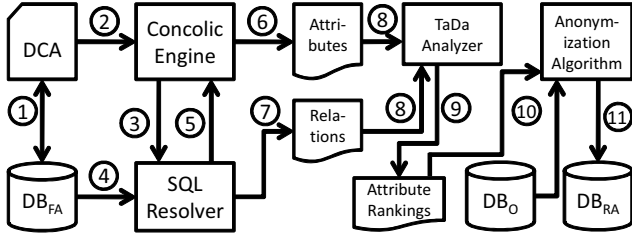


Figure 3. TaDa architecture and workflow. Solid arrows depict the flow of command and data between components, numbers indicate the sequence of operations in the workflow.

DCAs use their databases to which these algorithms are applied.

In order to determine how to maximize test coverage for DCAs while achieving desired privacy goals for databases that these DCAs use, we base our solution on three steps. First, we should determine how the values of attributes affect executions of statements in DCAs. Second, once it is clear what attributes affect statement executions, these attributes will be ranked using the number of statements and operations that these attributes affect.

The third step is to feed the ranked list of attributes into an anonymization algorithm that uses this information to determine how to proceed with applying anonymization techniques. Recall that business analysts, security experts, and database administrators identify attributes that contain sensitive information as well as QIs that can be used to disclose this sensitive information. In general, only a few small subsets of these QIs should be anonymized. The idea is to automatically select those attributes from a list of QIs whose values affect as few execution paths as possible.

Typically, control-flow decisions in the program are affected by a smaller subset of input data. 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 input data may not affect most branch conditions in DCAs, and therefore test coverage will not be affected much if this data is anonymized. Thus it is beneficial to focus anonymization on those aspects of the data that do not influence any or just few deeply nested control-flow decisions. To summarize, our key insight is that anonymizing a database attribute that does not influence control flow decisions has little impact on the DCA’s behavior induced by the database, in terms of statement coverage. For testing we use this insight to maintain higher test coverages when anonymizing databases.

## B. Architecture

Figure 3 shows the architecture of TaDa. The inputs to TaDa are the DCA bytecode and the fully anonymized database  $DB_{FA}$  (including its schema) that this DCA uses (1). TaDa first instruments the DCA automatically, by

inserting method calls (i.e., callbacks) before and after each instruction in the code. Test cases that drive DCA can be obtained from either existing cases from older versions of the same DCA, or automatically generated with concolic exploration [19]. The purpose of these test cases is to drive DCA for concolic engine to collect program behavior information. Thus lack of oracle is not a problem.

During DCA execution, the callbacks enable TaDa to create and maintain a precise symbolic representation of each element of the DCA’s execution state, including the invocation stack, operand stacks, local variables, and the heap (2). TaDa maintains the symbolic state during program execution, by mirroring the effect of each user program instruction in the symbolic state. This includes the effects of reading and writing any memory location, performing integer and floating point arithmetic, local and inter-procedural control-flow, and handling exceptions.

When the DCA accesses the database (1), TaDa tracks this access via the inserted instrumentation instructions. TaDa treats a value directly retrieved from a database differently than all other values, by “tainting” it in the symbolic state, i.e., by representing it with a symbolic variable instead of a symbolic literal. Any value derived from a database value is then represented as a symbolic expression that contains one or more symbolic variables. This dynamic taint analysis enables TaDa to determine how database values affect the control flow of DCAs.

When executing the DCA, TaDa captures its SQL query strings (3) and passes them to the SQL resolver, which uses an SQL parser and the database schema (4) to obtain  $\langle t, a \rangle$  pairs, where  $t$  is the table and  $a$  is the attribute of this table that is referenced in this SQL query. These pairs (5) are passed back to the concolic engine, which determines how these attributes are used by the DCA by tainting variables that hold the values of these attributes.

Since not all returned attributes are used in the DCA, the engine (6) outputs the list of attributes whose values affect some execution path(s) in the DCA. In addition, the SQL resolver (7) outputs the list of relations between tables (e.g., foreign keys and referential integrity constraints) that is obtained from the database schema SQL queries that the concolic engine passed to the SQL resolver. These attributes and relations are passed (8) to the TaDa analyzer. The analyzer ranks these attributes based on how many statements their values affect, and (9) the ranked list of attributed is outputted for review by security experts and business analysts as specified in the Steps 2-5 of the TaDa process.

The ranked list of attributes dictates (10) to the anonymization algorithm that given equal affect on a privacy policy, attributes with lower rank should be selected as QIs. The algorithm (10) applied anonymization techniques to the original database  $DB_O$ , taking it and the ranked list of attributes as the inputs. The algorithm outputs (11) the

resulting anonymized database  $DB_{RA}$ .

### C. Ranking Attributes

DCAs can access different attributes in their databases and use the values of these attributes in different ways. Some values 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.

To understand which attributes affect DCAs the most, we rank these attributes by counting the numbers of statements that their values affect. To do that we construct and traverse a control-flow graph (CFG) of the DCA. When traversing the CFG we count the number of statements that are directly 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, 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. Extending it to compute how many statements are affected by these attributes precisely is a subject of future work.

## IV. EXPERIMENTAL EVALUATION

In this section we describe the results of experimental evaluation of TaDa on four small example Java programs. What we investigate in this paper is whether it is possible for organizations to achieve both privacy and test coverage goals when applying  $k$ -anonymity. The core of our investigation is Tada, a tool that is used in test centers to link the attributes in the tables to variables in the DCAs to help security experts to select QIs that affect the DCAs the least.

### A. Research Questions

We seek to answer the following research questions.

- RQ1 How effectively does TaDa recommend selection of attributes as QIs? That is, is it possible to preserve test coverage while achieving a desired level of anonymity with TaDa when compared to the popular anonymization algorithm Datafly [20]?
- RQ2 Does using TaDa lead to a significant decrease of performance for applying anonymization techniques when compared to choices made by human experts?

DCA	App [kNCLOC]	Test	DB [MB]	Tbl	Att	IC %	FAC %
DurboDax	2.8	2.0	49	27	114	75	23
HealthCare	.8	.8	241	10	54	100	18
RiskIt	4.3	2.6	628	13	57	68	37
UnixUsage	2.8	.9	21	8	31	73	28

Table I

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, FAC = COVERAGE WITH THE FULLY ANONYMIZED DATA. WE COUNTED THE PERCENTAGE OF COVERED SQL STATEMENTS FOR THE APPLICATION HEALTHCARE.

### B. Subject Programs

We evaluate TaDa with four example 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 tests, and the presence of embedded SQL queries that these programs use.

We selected four subject programs that come with test cases. HealthCare is a program for maintaining health information records<sup>1</sup>. HealthCare differs from other subject DCAs in that its code does not contain SQL queries, which are supplied in a separate text file. Users can execute the application by supplying SQL queries as its inputs along with other data.

RiskIt is an insurance quote program<sup>2</sup>. DurboDax enables customer support centers to manage customer data<sup>3</sup>. Finally, UnixUsage is a program for obtaining statistics on how users interact with Unix systems using their commands<sup>4</sup>.

Table I 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 noncommented lines of code, NCLOC for the program code and accompanying tests. The source code of the project ranges from 0.8 to 4.3K NCLOC. The test cases range from 0.8 to 2.6K NCLOC. Other columns show the size of database, number of tables and attributes in these databases, test coverage that is obtained with the original database, and test coverage that is obtained with a fully anonymized database.

### C. Methodology

To evaluate TaDa, we carry out experiments to explore its effectiveness in enabling users to determine how to preserve test coverage while achieving data anonymity (RQ1), and how the overhead introduced by TaDa affects its use (RQ2).

<sup>1</sup><http://healthcaredbapp.sourceforge.net> as of March 5,2010.

<sup>2</sup><https://riskitinsurance.svn.sourceforge.net> as of March 5,2010.

<sup>3</sup><http://se547-durbodax.svn.sourceforge.net> as of March 5,2010.

<sup>4</sup><http://sourceforge.net/projects/se549unixusage> as of March 5,2010.

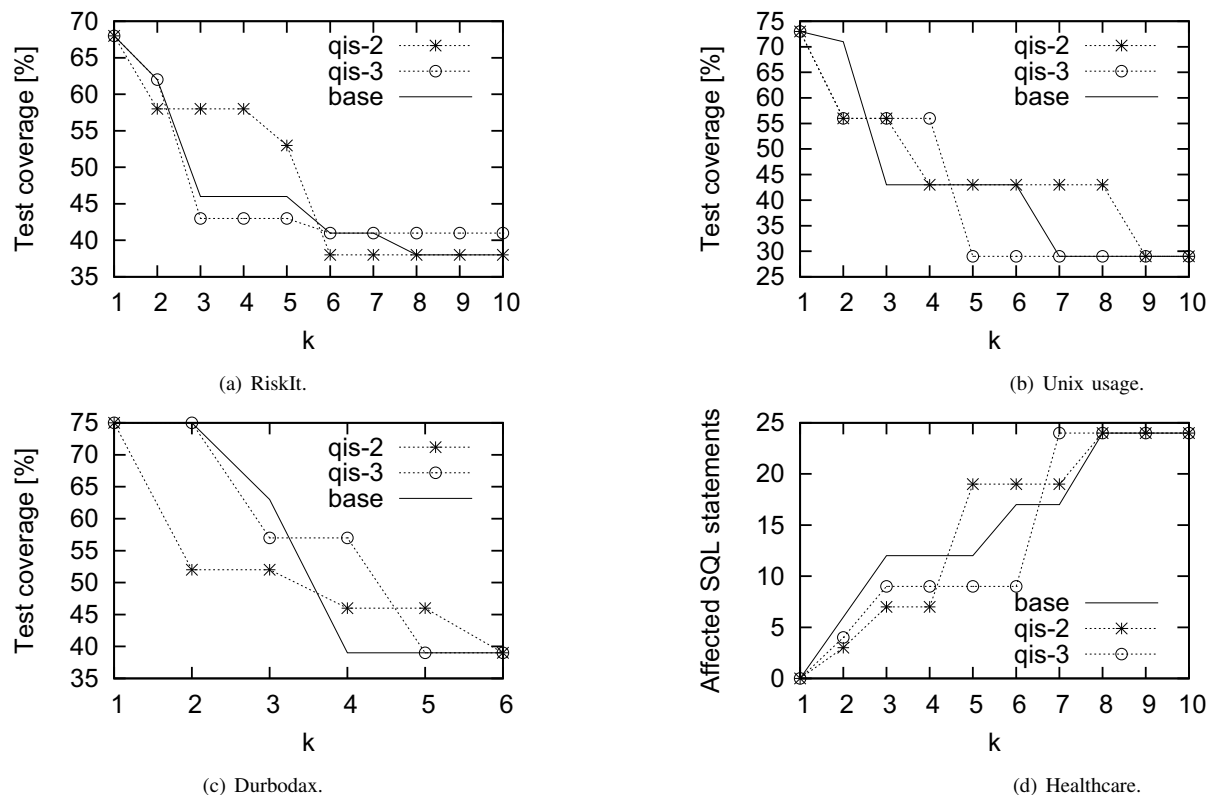


Figure 4. Experimental results for the subject DCAs for different sets of QIs (designated as `base`, `qis-2`, and `qis-3`), where the QI set `base` is the baseline obtained using the attribute selection heuristic of the algorithm Datafly.

1) *Privacy Goal*: We choose  $k$ -anonymity as the privacy goal for this experiment. Our choice is dictated by a few factors. First,  $k$ -anonymity is a widely used privacy goal. Second, algorithms to achieve  $k$ -anonymity are well-studied and understood. In addition,  $k$ -anonymity allows users to set different levels of privacy by changing the value of  $k$ , and it is important for our experiments since we attempt to find the balance between a sufficient level of privacy and test coverage. Finally, the same level of  $k$  can often be achieved by selecting different attributes for QIs, and it allows TaDa to make recommendations as to what attributes would affect test coverage the least.

2) *The Structure of the Experiment*: Unfortunately, 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 over  $10^{30}$  combinations for the subject databases. Given that databases of the subject DCAs contain between 31 and 114 attributes, it is challenging to select a subset of them as QIs while preserving a higher level of test coverage. Recall that in the industry this job is done by a team that comprises business analysts and database experts. Therefore we conduct this experiment only with three sets of QIs: one that is selected using the heuristics for a well-known and frequently used

$k$ -anonymity algorithm and the other two sets of QIs are selected using recommendations of TaDa from the ranked list of attributes. We selected two sets of QIs since there is more than one set of QIs to protect sensitive information, and we wanted to obtain more measurements to evaluate TaDa. Each set of QIs used from 17 to 36 attributes. We analyze and compare results of this experiment to answer RQ1 and RQ2.

A baseline for our experiment is applying a  $k$ -anonymity algorithm called Datafly [20] to the databases of the subject DCAs. The input to this algorithm is a list of attributes sorted in the descending order by the numbers of distinct values, and generalization hierarchies for these attributes (i.e., it is an accepted heuristic for selecting attributes as QIs). An example of a generalization hierarchy is replacing values of exact addresses with names of cities, which in turn can be generalized by replacing them with names of counties and states. When Datafly is applied to subject databases, it selects attributes from the input list and applies certain levels from the generalization hierarchies to suppress and generalize data. The algorithm runs until there are more than  $k$  undistinguishable records in the database for every entity. Once the databases are anonymized, we run test cases and measure the resulting test coverage for different values of  $k$ . These values are reported in Figure 4 with lines marked as

base, which is the baseline for this experiment.

To evaluate how well TaDa recommends the selection of attributes for QIs we include the recommended attributes on the list of QIs for the algorithm Datafly. Consider for a different baseline that after fully anonymizing all data in the databases, the coverage dropped between 31% and some 80% as it is shown in Table I. Unlike the selection heuristics for Datafly, TaDa recommends attributes for QI based on their measured impact on executing statements of their corresponding DCAs. That is, attributes that affect the smallest number of statements will be on the top of the list; conversely, attributes that impact most statements will be on the bottom. With lower values of  $k$ , executing Datafly will affect fewer QIs and involve lower levels of generalization hierarchies; conversely, increasing values of  $k$  will lead to anonymizing more QIs thereby affecting test coverage. We observe and measure a number of response variables and report them in Section IV-E.

3) *Variables*: Main independent variables are the values of  $k$  in  $k$ -anonymity and the selection of different sets of QIs from database attributes. Two main response variables are the time 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. For the DCA HealthCare, we measure the number of affected SQL queries, since the program logic is embedded in these queries that are supplied with the application. In addition, we measure time and memory consumption of running TaDa's concolic engines on the subject DCAs.

4) *Factors Outside Our Control*: There are three factors that make experimental evaluation difficult. First, anonymization time is a function of the implementation of anonymization techniques as well as the structure of the subject database. TaDa does not address the issue of how to improve existing anonymization techniques, it uses them as tools to achieve the desired levels of anonymity.

Second factor is that the structure of the database and semantic relations between data may limit choices of QIs and thus reduce the effectiveness of the recommendations of TaDa. In the worse possible case, to achieve the level of  $k = 2$  selection of all attributes as QIs may be required, leaving little choice for DCA owners.

In a similar vein, the logic of DCAs may leave little choice for DCA owners. If values of some attribute control a branch condition that affects a large number of statements, and this attribute must be selected as a QI, then the best thing that TaDa can do is to advise the DCA owner on the possibly lost level of test coverage. In the end it is the decision of the DCA owner on how to handle the trade-off between test coverage and data privacy.

#### D. Threats to Validity

A threat to the validity of this experimental evaluation is that our subject programs are of small 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 tables and attributes may have different characteristics compared to our small 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 TaDa. Making TaDa more scalable is part of our future work.

Another threat to the validity is that we selected attributes whose values must be protected from the set database attributes. That is, for these initial experiments we played the role of the business analyst and database expert team ourselves. Selecting these attribute as sources of sensitive information may affect further selection of other attributes as QIs. Since the domains of the subjects DCAs are easy to understand, we believe that our selections are close to what a group of privacy experts could select. Even though we cannot say with certainty that others would select the same attributes, we believe that even if others were selected, it would not change the results of our experiments drastically.

Additional threats to validity of this study is that in our experimentation we use programmers who created unit tests to drive DCAs, and this task should be tackled by testing centers. Specifically, the bad unit tests that do not exercise paths that lead to launching SQL queries can cause TaDa to miss attributes that affect DCA coverage. Currently, we do not provide any support for helping programmers to create right unit tests or select input data, it is a subject of our future work.

#### E. Results

The results of these experiments are shown in Figure 4. This figure includes four graphs, one for each subject DCA. Graphs in Figure 4(a)–Figure 4(c) show how test coverages of the DCAs Riskit, UnixUsage, and DurboDax depend on the values of  $k$  when applying Datafly to databases, and the graph in Figure 4(d) shows how many SQL statements are affected when executing the DCA Healthcare that uses these statements to retrieve data from its database. Graphs in Figure 4(a)–Figure 4(c) show that test coverage drops when increasing the values of  $k$  (i.e., increasing the level of protection) and graphs in Figure 4(d) show that the number of affected SQL statements grow.

These graphs give us an insight into how selection of attributes as QI affects drops in test coverage. Recall that the graph marked as `base` represents a baseline obtained by applying the algorithm Datafly without using TaDa recommendation. In every case we see that TaDa issued recommendations that permitted higher values of test coverage for certain values of  $k$ . For example, it is shown in Figure 4(a) that for  $3 \leq k < 6$  TaDa recommended QIs `qis-2` beat the baseline test coverage by approximately more than 11%.

DCA	Min2 [mins]	Max2	Min3 [mins]	Max3	Dif2 %	Dif3 %
DurboDax	-2	0.01	0	1	33	13
HealthCare	-23	39	-128	2	19	58
RiskIt	-91	4	-207	2	29	43
UnixUsage	-0.1	25	-0.1	21	130	103

Table II

ANONYMIZATION TIMES FOR DIFFERENT SUBSETS OF QIS WHEN COMPARED TO THE BASELINES THAT USE DATAFLY. POSITIVE VALUES MEAN THAT IT TAKES ADDITIONAL TIME AND NEGATIVE VALUES MEAN THAT IT TAKES LESS TIME TO ANONYMIZE DATA WHEN COMPARED TO THE BASELINE TIMES. MIN2 = THE MINIMUM AMOUNT OF TIME FOR QIS-2, MAX2 = THE MAXIMUM AMOUNT OF TIME FOR QIS-2, MIN3 = THE MINIMUM AMOUNT OF TIME FOR QIS-3, MAX3 = THE MAXIMUM AMOUNT OF TIME FOR QIS-3, DIF2 = MAX PERCENTAGE DIFFERENCE FOR QIS-2, DIF3 = MAX PERCENTAGE DIFFERENCE FOR QIS-3.

Interestingly, with all subject DCAs the test coverages dropped to their worst levels when  $k > 7$ . We confirmed this phenomenon with other research groups in Accenture Technology Labs whose researchers worked independently on applying data anonymization algorithms to different databases. To explain this result, consider the DCA RiskIt (see Figure 4(a)) where TaDa recommends users to avoid anonymizing attributes `age` and `occupationcode` because their values are used in many branch conditions and consequently affect a large number of statements. Anonymizing values of these attributes can be easily avoided when achieving the level of  $k$ -anonymity up to five. Less impactful attributes are anonymized, and the coverage of RiskIt drops only to little more than 90% of the original test coverage. However, when trying to achieve the level of  $5 < k \leq 8$ , the values of both attributes have to be anonymized, and subsequently test coverage drops to little over 60% of the original test coverage. To summarize, higher levels of  $k$  require wider applications of anonymization techniques to QIs that invariably affects test coverage of DCAs.

While the differences are big in resulting test coverages for difference values of  $k$ , time differences are not that significant between anonymizing different sets of QIs. Table II contains the results of the anonymization times when compared between using QIs that were recommended by TaDa with the baseline results of the algorithm Datafly. The maximum increase in anonymization time is a little more than 39 minutes, which is acceptable for such a difficult and laborious process as test outsourcing. At the same time, TaDa selection recommendations led to decreases in anonymization time at 117 minutes.

Replacing one attribute with some other for a choice of QI may increase the time it takes to anonymize data. However, the intuition is that it should not happen. Recall that the heuristic that guides the QI selection process lead to

attributes with many distinct values or big ranges in values. For example, attributes as `age` and `occupationcode` do have many distinct values, they both can be used to identify sensitive information. However, replacing `age` with `occupationcode` and vice versa for a choice of QI is unlikely to change the running time of data anonymization process, since it involves physical data scrambling, which is proportional to the size of data.

## V. RELATED WORK

Our work is related to regression testing since TaDa 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. Some of the popular regression testing techniques include analyzing the program’s control-flow structure, analyzing changes in functions, types, variables, and macro definitions [21], using def-use chains [22], and constructing procedure dependence graphs [23], and analyzing code and class hierarchy for object-oriented programs [24]. 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 TaDa is an anonymization technique for protecting private information in bug reports that are delivered to vendors when programs crash on computers of customers [25]. 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. Like TaDa, this technique uses symbolic execution to create new inputs that allow vendors to reproduce the bug while revealing less private information than existing approaches. A key difference is that TaDa works with DCAs which use databases, and it is unclear if this technique can be extended to anonymize databases to preserve test coverage.

The concolic engine of TaDa builds on the pioneering work of Godefroid et al. and Cadar et al. on the Dart and EGT concolic exploration systems [26]. This first generation of tools, which also includes Cute and jCute [19], showed the feasibility of dynamic symbolic (concolic) execution, but only supported a subset of user programs. The concolic engine of TaDa belongs to the second generation of tools [27], [28], [29], which aim at supporting any user program written in a given programming language.

## VI. CONCLUSION

In this paper we show that applying a popular data privacy approach called  $k$ -anonymity leads to serious degradation of test coverage. We offer a novel and effective approach called TaDa that helps organizations to understand how to balance privacy and software testing goals. We show that for small

values of  $k \leq 6$  it is possible to achieve a higher test coverage with our approach than with the current state-of-the-art  $k$ -anonymization algorithm, Datafly. However, we also show that for higher values of  $k \geq 7$  test coverage drops to less than 30% from the original coverage of more than 70%, thus making it difficult to achieve good quality when testing DCAs while applying data privacy.

#### REFERENCES

- [1] G. M. Kapfhammer and M. L. Soffa, "A family of test adequacy criteria for database-driven applications," in *Proc. 11th ACM SIGSOFT FSE*. ACM, 2003, pp. 98–107.
- [2] "International data privacy laws," <http://www.informationshield.com/intprivacylaws.html>.
- [3] D. J. Richardson, S. Leif-Aha, and T. O. O'Malley, "Specification-based Test Oracles for Reactive Systems," in *Proceedings of the 14th ICSE*, May 1992, pp. 105–118.
- [4] C. Kaner, J. Bach, and B. Pettichord, *Lessons Learned in Software Testing*. New York, NY, USA: John Wiley & Sons, Inc., 2001.
- [5] T. E. Murphy, "Managing test data for maximum productivity," [http://www.gartner.com/DisplayDocument?doc\\_cd=163662&ref=g\\_economy\\_2reduce](http://www.gartner.com/DisplayDocument?doc_cd=163662&ref=g_economy_2reduce), Dec. 2008.
- [6] A. S. Namin and J. H. Andrews, "The influence of size and coverage on test suite effectiveness," in *Proc. 18th ISSTA*. ACM, 2009, pp. 57–68.
- [7] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, 1997.
- [8] J. T. Alexander, M. Davern, and B. Stevenson, "Inaccurate age and sex data in the census pums files: Evidence and implications," National Bureau of Economic Research, Working Paper 15703, January 2010. [Online]. Available: <http://www.nber.org/papers/w15703>
- [9] C. Bialik, "Census bureau obscured personal data – too well, some say," *The Wall Street Journal*, Feb. 2010.
- [10] W. Aspray, F. Mayades, and M. Vardi, *Globalization and Offshoring of Software*. ACM, 2006.
- [11] Datamonitor, "Application testing services: global market forecast model," Aug. 2007.
- [12] C. C. Aggarwal and P. S. Yu, *Privacy-Preserving Data Mining: Models and Algorithms*. Springer, 2008.
- [13] P. Samarati, "Protecting respondents' identities in microdata release," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 6, pp. 1010–1027, 2001.
- [14] INCITS/ISO/IEC, "Information technology - database languages - sql - part 5: Host language bindings (sql/bindings)." INCITS/ISO/IEC, Tech. Rep., 1999.
- [15] G. Cormode and D. Srivastava, "Anonymized data: generation, models, usage," in *Proc. 35th ACM SIGMOD*. ACM, 2009, pp. 1015–1018.
- [16] W. Landi, "Undecidability of static analysis," *ACM Lett. Program. Lang. Syst.*, vol. 1, no. 4, pp. 323–337, 1992.
- [17] J. Nin, J. Herranz, and V. Torra, "Attribute selection in multivariate microaggregation," in *Proc. PAIS*. ACM, 2008, pp. 51–60.
- [18] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Mondrian multidimensional k-anonymity," in *Proc. 22nd ICDE*. IEEE, 2006, p. 25.
- [19] K. Sen and G. Agha, "Cute and jCute: Concolic unit testing and explicit path model-checking tools," in *Proc. CAV*. Springer, Aug. 2006, pp. 419–423.
- [20] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.
- [21] J.-M. Kim and A. A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *ICSE*, 2002, pp. 119–129.
- [22] M. J. Harrold, R. Gupta, and M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Trans. Softw. Eng. Methodol.*, vol. 2, no. 3, pp. 270–285, Jul. 1993.
- [23] R. A. Santelices, P. K. Chittimalli, T. Apiwattanapong, A. Orso, and M. J. Harrold, "Test-suite augmentation for evolving software," in *ASE*, 2008, pp. 218–227.
- [24] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: a tool for change impact analysis of java programs," in *OOPSLA*, 2004, pp. 432–448.
- [25] M. Castro, M. Costa, and J.-P. Martin, "Better bug reporting with better privacy," in *Proc. 13th ASPLOS*. ACM, Mar. 2008, pp. 319–328. [Online]. Available: <http://research.microsoft.com/jpmartin/papers/-Castro08Better.pdf>
- [26] P. Godefroid, N. Klarlund, and K. Sen, "Dart: Directed automated random testing," in *Proc. ACM SIGPLAN PLDI*. ACM, Jun. 2005, pp. 213–223.
- [27] N. Tillmann and J. de Halleux, "Pex - white box test generation for .Net," in *Proc. 2nd International Conference on Tests And Proofs (TAP)*. Springer, Apr. 2008, pp. 134–153.
- [28] C. Cadar, D. Dunbar, and D. R. Engler, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *Proc. 8th USENIX OSDI*. USENIX, Dec. 2008, pp. 209–224.
- [29] C. Csallner, N. Tillmann, and Y. Smaragdakis, "DySy: Dynamic symbolic execution for invariant inference," in *Proc. 30th ACM/IEEE ICSE*. ACM, May 2008, pp. 281–290.