SEMANTIC BASED TEST DATA EXTRACTION FOR INTEGRATED SYSTEMS (iSTDE)

Ali Raza and Stephen Clyde
Computer Sceince Department, Utah State University, 400 Old Main Hill, Logan, Utah 84322-1400, U.S.A.

Keywords: Test data extraction, Test data generation, Testing database applications, Semantic-based data extraction, Integrated systems, Integrated healthcare systems.

Abstract: Testing an integrated information system that relies on data from multiple sources can be a serious challenge, particularly when the data is confidential. Such is the case for the Child-Health Advanced Record Management (CHARM) system, which is now in production at the Utah Department of Health. CHARM allows various public health-care programs, like vital records, immunization, and hearing screening, to seamlessly access data from each others' databases in real-time. Since CHARM deals with confidential health-care information, it was impossible to use real data for testing purposes, especially since the development and testing environments were outside the confidential environment in which CHARM operates. This paper describes a test-data extraction tool built and successfully used for testing the CHARM system. This tool, called Semantic based Test Data Extractor for Integrated Systems or iSTDE, reads a consistent cross-section of data from the production databases, manipulates that data to obscure individual identities while preserving overall data characteristics that are critical to thorough system testing, and finally moves that test data from the confidential production environment to the unprotected test environment.

1 INTRODUCTION

Child Health Advanced Record Management (CHARM) is an integrated system that provides health-care professionals with accurate and timely information about children in Utah whose medical records are housed in various federated public healthcare databases, including Vital Records (VR), the Utah State-wide Immunizations Information System (USIIS), and Early Hearing Detection and Invention (HiTrack).

A collaborative team of software engineers from Utah State University (USU), Utah Department of Health (UDOH), and Multimedia Data Services Corporation (MDSC) started developing CHARM in November 2000. Its architecture, illustrated in Figure 1, is that of an arms-length information broker (Clyde and Salkowitz, 2006) with

- A CHARM server, which is the information broker, and
- A CHARM agent for each connected database also called a participating program or PP.

When a user of a PP requires CHARM-accessible data, the PP submits a request for that data to CHARM via its own agent. That agent is responsible for mapping PP-specific data types and identifiers to CHARM-specific data types and identifiers. It next passes the modified query onto the CHARM server. The CHARM server either

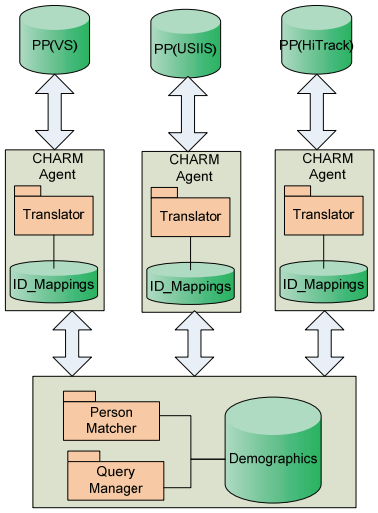

Figure 1: CHARM Architecture.

Raza A. and Clyde S. (2010).
SEMANTIC BASED TEST DATA EXTRACTION FOR INTEGRATED SYSTEMS (iSTDE).
In *Proceedings of the Third International Conference on Health Informatics*, pages 173-180
DOI: 10.5220/0002692401730180

looks up or computes an appropriate strategy for processing the query and then executes that strategy. This process may involve retrieving information from several other PPs via their CHARM agents and merging the results of those individual data retrievals into a final query result.

The first functional prototype was successfully demonstrated in March 2002. It was at this point that the developers began to see the real challenges of testing an integrated system that involves confidential data. With the three original participating programs, the system made use of seven different databases: three from the participating programs, three used by the agents to map PP-specific IDs to internal CHARM IDs, and one used by the CHARM server to match and link persons based on their demographic information (CHARM). From a testing stand-point, however, such data separation made generating realistic test data difficult. At first, the developers tried to create test data by hand. This quickly proved to be time consuming and error prone. Next, the developers built an automated test-data generator that created test data for each database using that database's scheme and codified knowledge about field domains, constraints, and overall data characteristics (Maddy, 2006). Such an approach allowed the developers to create large amounts of test data, but correlating the information between different databases and creating patterns similar to those in the real data proved difficult.

So, for the latest version of CHARM, the developers have taken a new approach for creating test data. Specifically, they created a distributed tool, called Semantic Test Data Extractor for Integrated Systems (iSTDE), which first extracts a consistent cross-section of data from the production databases. It next manipulates that data in a way that obscures individual identities, while preserving other important aggregate data characteristics, such as the frequency of name occurrences, the percentage of multiple births (i.e., twins), and the presence of bad data. Preserving these characteristic is critical to effective system testing of components like a person matcher. After de-identifying the test data, iSTDE moves that test data from the production environment to a test environment.

Section 2 discusses related work for creating realistic test data, while, Section 3 provides some additional background on the production and test environment of testing CHARM. Section 4 describes the process iSTDE uses to extract data, de-identify that data, and move it to a test environment. Experience and observations in using iSTDE are presented in Section 5. Future work is in Section 6.

2 RELATED WORK

In general, approaches for test data creation fall in two general categories: one based on automatic generation and the other based on real data extraction.

A review of eight automated test-data generation tools revealed six different common techniques for generating data at a field level, i.e., for a domain. See Table 1 for list of the tools reviewed and Table 2 for the techniques each supported.

The first two techniques create random data based on a field's data type along with some simple constraints. For example, an algorithm based on random generation could populate a salary field in a payroll table with values between $20,000 and $65,000. Similarly, a random-generation algorithm could populate a first name field in a person table with a string between 1-10 characters long, containing characters A-Z. In general, random generation is more applicable to numeric fields than other types of domains. Six of the eight tools support random generation for numeric data, while only three support it for strings.

The third technique constrains the random generation of data by percentages that represent value distributions in real data. For example, imagine a person table with 20% of the records having birth dates in 2008 and the remaining 80% in 2007. A tool that supports this type of data creation could preserve such distributions. Only one of the tools supports this type of random data generation.

The fourth technique generates data according to user-defined grammars. For example, the grammar *Aa-9999* could generate data that has one capital letter, followed by one small letter, a dash and four numeric digits. This technique is most applicable for string domains with an implicit language that can be easily defined with a pattern or simple grammar. Interestingly, it is common for database schemes to have fields with simple hidden languages, but only two of the eight tools support this technique.

The fifth technique pulls randomly selected data from a pre-defined domain. For example, this technique could be used to populate a last-name field from a pre-defined domain of common Spanish names. Four of the eight tools support this technique, and several of them even had some built-in domains for female names, male names, countries, etc.

The sixth technique identifies an algorithm that links child records to parent records in hierarchical structures. For example, an algorithm that uses this

technique could be used to generate data for a purchasing system consisting of customer, order, line item tables that relate to each other via referential integrity constrains.

Although automated test-data generation techniques can save time compared to collecting and loading meaningful test data by hand, they fall short of producing test data that possess many of characteristics found in real data, such as:

- the presence or frequency of missing values;
- the presence or frequency of incomplete information;
- the presence of garbage data;
- duplicates, wherein the duplicates were caused by or allowed to exist because of other field values; and
- other characteristics caused inter-field dependencies.

The second approach, test-data extraction, attempts to create test beds from real data sources. A review of the eight tools uncovered extraction techniques at three different levels, namely, extracting data from a single file, extracting data from multiple tables in a single database, and extracting data from multiple unrelated databases. See the first three rows in Table 3.

Three of the tools support the first technique, which has some similarities to test-data generation from predefined domains. However, a key difference is that test-data extraction can produce test data with realistic characteristics without explicitly having to state those characteristics.

The second technique deals with extracting test data from multiple tables in a real database. This type of test-data extraction does everything supported by the first technique, but it also maintains inter-record dependencies across the tables. However, these dependencies can go beyond the referential integrity constraints mentioned above. Specifically, they can include frequency constraints involving fields from multiple tables. Three tools support this type of data extraction, at least to some degree.

The third technique, which only one of the eight reviewed tools supports, goes a step further by allowing users to extract data from multiple databases. However, without any cross-correlation of data between the databases, this technique can be viewed as simply a convenience for performing multiple, separate extractions.

Clearly, being able to create test data for multiple databases is necessary for testing integrated systems, but it is not enough. To test an integrated system, its constituent components (i.e., participating information systems) need realistic and correlated slices of data that contain the same inter-relationships and hidden dependencies from the production databases. For example, it would be meaningless to extract one set of person records from one database and a non-overlapping set of records from another database. Testing would not be able to verify the results of any actual data integration.

Also, to test integrated systems that contain confidential data, it is important to remove or hide all identified personal information so that testing can be conducted in unsecured environments.

To address the need for correlating data across databases and for de-identified test data, we have added two additional test-data extraction techniques to Table 3, namely, correlated real data from multiple related databases and de-identified data from confidential databases. The iSDTE tools presented in the next section support these additional techniques.

3 ISTDE ENVIRONMENT

In general, multiple CHARM execution environments exist, including one for production, one for staging and user-acceptance testing, several for system testing, and at least six for development. From a data-security perspective, these environments can be grouped in two categories: confidential and unprotected. The confidential environments, which include the production environment and staging environment, are protected by firewalls in UDOH. Only authorized users can access these environments containing sensitive demographic and health-care data. The unprotected environments, which include all the system testing and development environments, run on a variety of machines and networks outside of UDOH firewalls, and may be used by individuals not authorized to see real data.

Besides the access restrictions, the confidential and unprotected environments differ in terms of the database managers they use for the various data sources. The data sources in the confidential environments are either the actual production databases or staging databases for the production system. In either case, these databases are tied to legacy software and, therefore, rely on a number of different database managers, including Oracle, Microsoft SQLServer, Postgres, and Pervasive.

Table 1: Eight software packages reviewed.

Abr.	Software Package	Author / Vendor
DG	GenerateData.com (GenerateData.com,2008)	GenerateData.com
SE	DTM Data Generator (SqlEdit, 2008)	DTM Soft
FS	ForSQL Data Generator (www.forsql.com, 2008)	ForSQL
TS	Automated Test Data Generator (http://www.tethyssolutions.com/T10.htm, 2008)	Tethys Solutions
DN	DB Data Generator V2 (www.datanamic.com, 2008)	Datanamic
TB	TurboData (www.turbodata.ca, 2008)	Turbo Computer Systems, Inc.
TN	Tnsgen – Test Data Generator(www.tns-soft.com , 2008)	TNS Software Inc.
EM	EMS Data Generator for MySQL (http://www.sqlmanager.net/en/products/postgresql/datagenerator)	EMS Inc.

Table 2: Six common test-data generation features

	Test Data Generation Features	DG	SE	FS	TS	DN	TB	TN	EM
1	Random numeric data generation		●	●	●	●	●	●	
2	Random string data generation				●		●	●	
3	Percentage-based data generation		●						
4	Generate data from user-defined grammars		●			●			
5	Generate data from predefined domains	●		●		●	●		
6	Generate data for database, with master child relations						●		

Table 3: Test-data extraction techniques.

	Test Data Extraction	DG	SE	FS	TS	DN	TB	TN	EM
1	From real data from files		●	●				●	
2	From real data from one database		●	●					●
3	From uncorrelated real data from multiple databases								●
4	Correlated real data from multiple related databases								
5	De-identified data from confidential databases								

All the unprotected environments, on the other hand, use Postgres as the database manager to eliminate extra licensing fees that might otherwise be necessary. However, using a different database manager for testing introduces two new challenges. First, the types of database that the integration system accesses will depend on the environment it is running in. So, testing iSTDE in one of the unprotected environments may not verify the correctness of the database drivers, connection strings, or SQL-statement syntax. For CHARM, we solved this problem by doing a final system test in the staging environment, which does use all of the same types of databases as the production environment.

Second, converting all the data to Postgres for the unprotected environment introduces certain data-type mapping problems. Some data types in the original database do not have compatible data types in Postgres. For example, SqlServer supports a global unique identifier (GUID) data type that Postgres does not support. So, iSDTE maps SqlServer GUIDs to an alternative data type, like VARCHAR. Section 4.2 describes iSDTE's solution to this problem in detail.

4 APPROACH

The iSTDE software itself is installed in a environment, thereby ensuring that no unauthorized person can execute it. When iSTDE executes, it goes through seven steps to create a consistent set of de-identified test data from the confidential data and then moves it to an unprotected environment. See Figure 2. Each of these steps is described in more detail below.

4.1 Specifying Extraction Parameters

In the first step, a user specifies what data to extract (e.g., all children born from 7/1/2008 to 9/30/2008) and the target environment wherein test data should ultimately be sent, along with a username and password for accessing that environment. In addition, the user can specify the location of a temporary database within the confidential environment that iSTDE will use to collect and manipulate the test data before sending it over to the target environment.

iSTDE also supports a number of other configuration parameters that the user typically does not change, such as connection strings for the various data sources in the confidential environment. These parameters are kept in a properties file and only need to be changed if the data sources in the confidential environment change.

4.2 Creation of Temporary Databases

The second step in the iSTDE execution involves creating the temporary database in a confidential environment to hold the extracted data from multiple source databases, while they are being collected and manipulated. In this step, iSTDE first makes sure that there are no existing temporary databases in confidential environment. It then retrieves schema metadata for all the source databases and transforms them into Postgres creation scripts. Next, it executes those scripts to create the temporary database, with all of the necessary tables, indices, views, stored procedures, and triggers. Further into the process in Step 6, iSTDE drops the temporary databases, so unnecessary copies of the extracted data are not left lying around.

iSTDE uses Postgres for the temporary database because it is Open Source and it supports a broad range of features and data types. Nevertheless, it does not support everything; nor did we find an Open Source database manager that did.

One challenge for Step 2 was accessing metadata. Some source databases do not allow external processes to read the database's metadata, or they do not support reflection. So, iSTDE could not automatically retrieve and analyze their structures directly. For such databases, iSTDE reads the metadata from an externally managed meta-data repository. This repository has to be updated manually when the real database's structure changes.

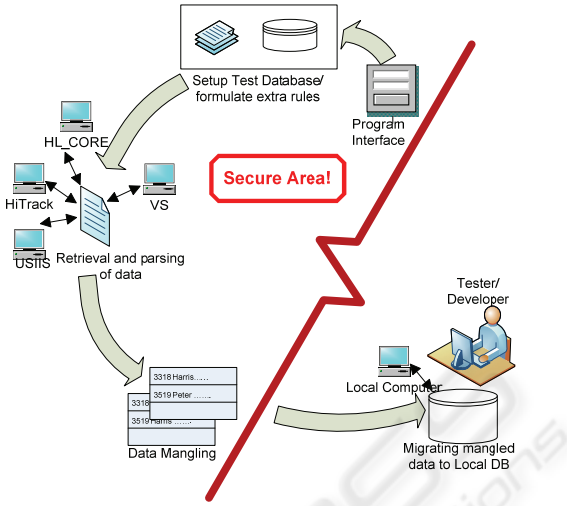

Figure 2: iSTDE Overview.

Another challenge was unsupported or incompatible data types, as mentioned in Section 3. For each unsupported data type, iSDTE designers selected an alternative Postgres data type and wrote a mapping function that converts data from the original type to the alternate type. So, when iSDTE comes upon a field with an unsupported data-type, it simply looks up its alternative data type and uses that type in the table creation scripts for the temporary database. Then, later in Step 3, iSDTE uses the corresponding mapping function to convert the values from that field before placing them into the temporary database.

The third challenge was handling views in iSTDE. The test databases need to support or simulate any views in the real database such that the legacy programs and integrated system will function correctly. There are two approaches for supporting a view. The first is to include all the tables and their data that makes up the view in the test set. However, this can be problematic because some views are very complex and may end up requiring far more data to be extracted into the test database than necessary. A second approach is to implement the views as tables populated with a snapshot (or a portion of a snapshot) of the view. This approach can reduce both the amount space required for the test data and the extraction time. However, this approach is only appropriate if the integrated system does not need to update any of the data involved in the view.

Like views, database procedures also need to be either implemented directly in the test database or simulated through tables, since different database managers use different procedural languages and it is very difficult to automate their extraction and

direct implementation. However, as with views, when the integrated system does need to modify the underlying data, simulating a stored procedure using a table of stored results is relatively straightforward. When this was not possible for CHARM, a programmer manually created versions of the store procedure in plsql (Postgres's procedural language.) Manual conversions of stored procedures need only be done once. After that, iSTDE can re-use them whenever needed.

A final challenge stems from version conflicts. The systems that comprise an integrated system, as well as the integration framework itself, evolve independent of each other. Changes do not occur in a lock-step chronology. One system will upgrade its database, while other systems are still using older structures. For example, over the past few years, there have been two major versions of the CHARM integration framework, and at least one significant database change to each of the participating programs. The database schemas for these versions have slightly different metadata. Such was the case when the CHARM developers were testing Version 2, yet the production environment was still using Version 1 data structures. Mapping data across versions of an integrated system is a significant problem. iSTDE handles this by adding some additional metadata to the external metadata repository, so it can track the version and then re-map data if necessary.

Some challenges are still unresolved. For example, in cases wherein iSTDE has to store the meta-data for a system in an external repository, changes to the original database's structure can create an inconsistency. Organization procedures have to be put in place and followed to ensure that changes to a participating information system are reflected in iSTDE metadata for that system. It would be better if more of this process could be automated or at least monitored by iSTDE.

4.3 Extraction and Loading of Real Data to Temporary Databases

The previous steps created empty temporary databases for holding the extracted data, with all the necessary tables, constraints, views (or their simulations), and stored procedures (or their simulations). Now in this the third step, iSTDE extracts a consistent slice of real data from the participating data sources in the confidential environment and loads that data into these temporary databases.

The process of extracting a data slice starts when iSTDE generates SQL queries through parsing and analyzing user-specified test-data selection criteria, e.g., child birth date range. One SQL query is generated for each of the relevant production databases. A challenge in data extraction was to ensure that the slice contains records for the same sample population across all of the participating programs. To ensure the slice's internal consistency, iSTDE uses cross-database links created and maintained by the integrated system. In CHARM, each agent maintains a mapping of its participating program's IDs to a common, internal CHARM ID. Together, these maps link the records for a person across all of the participating information systems. iSTDE uses and preserves these inter-database links to guarantee that the overall test data are internally consistent.

When these SQL queries return result sets, iSTDE uses the data in these result sets to construct SQL insert statements. While constructing these SQL insert statements, individual data fields in a result set are parsed according to the temporary databases' metadata. Later these insert statements are written to data files located in a confidential environment. The purpose of generating data files is to effectively utilize the connection time on production databases.

Once the process of extracting data into data files is complete, iSTDE loads these files to the temporary databases. A challenge was to load data in such a way as to not violate referential integrity constraints. iSTDE deals with this challenge by loading data files for parent tables before child tables. However, the cyclic nature of relational interdependencies among tables makes this solution unfeasible. A better approach would be to first load the data into tables and later implement referential integrity constraints.

4.4 Data Mangling

Once real data has been extracted and loaded into the temporary databases, iSTDE obfuscates that data by applying data mangling to each domain that contains personal indentifying information (PII). In this step, data mangling randomly swaps data values in the domain so the PII's of any given record are unrecognizable and untraceable, but without changing the overall characteristics of the data set. As mentioned early, preserving the overall characteristics of the data set is critical for thorough testing in integrated systems.

The first step of data mangling is the identification of PII domains, e.g., first names, last names, birth dates, addresses, etc. PII domains can be simple or composite. Simple domain consists of

only one identifying element, and composite domains may involve multiple elements. For example, male first name and phone number are simple domains, whereas a full address domain (street, city, state, zip code) may be a composite domain. To preserve consistency, composite domains have to be mangled as a whole unit. For example one instance of an address may be swapped with another random but complete address. iSTDE also sometimes subdivides a domain wherein swapping needs to be constrained by the value of some other element. For example, it partitions gender-dependent domains into female and male subset, i.e., first name domain is partitioned into male first names and female first names.

After PII domains selection, we build dictionaries for these domains. These domain dictionaries are data structures that consist of real domains and test domains. Real domains are data slices that are built from similar PII domains from across all databases included in CHARM, not just one database. For example, in the case of first male names, the dictionary real domain contains all first male name entries that exist in all tables of temporary databases. Test domains are populated by semi-random shuffling of real domain entries. The term semi-random hints that there are chances that an entry in a real domain maps to the same entry in a test domain. Entries in test domains would be the newly assigned values for the real data. In the next step of the mangling process, we swap all the values of PII domains in real data with the newly assigned test values, using domain dictionaries that provides mapping from real values to test values. Once we have mangled all the data, we then delete these dictionaries so that no one can perform reverse mapping to real data. Essentially, iSTDE deals with three different types of data mangling.

The first type is 1-1 logical domain dependency. Two domains are said to be logically dependent when they are semantically related to each other, a change in one domain requires a similar change in the other. Consider two domains, D1 and D2, which have a 1-to-1 logical dependency between them but have different data representations. When we swap a value in one of the domains, a corresponding swap must also be made in the second. More specifically, if x, x'$\in$ D1 and y, y'$\in$ D2 such that x $\leftrightarrow$ y, and x' $\leftrightarrow$ y', then if x is swapped with x', y must also be swapped with y' and vice versa. For example, consider two tables containing identical demographic information about patients. One table uses just one column to store birth dates, i.e., say 05/11/2009 for patient A, while another table uses three columns to store the same birth date of patient A, i.e., say 05 as MM, 11 as DD, and 2009 as YYYY. iSTDE ensures that two tables maintain the same logical dependency after mangling, that is, if the birth date 05/11/2009 is swapped with some other date 07/10/2007 in one table, iSTDE also makes the same logical swap in the other table that uses three columns to represent the birth dates.

The second type of dependency in the iSTDE mangling process is called data value dependency. Two domains D1 and D2 are said to have a data value dependency when for any single record that uses values from both domains, there is a constraint involving those values in these domains. Then, if values in D1 are swapped, a random swap must also be made in D2, but the original constraint must still hold (if the original record satisfies that constraint.) More specifically, if x, x'$\in$ D1and y, y'$\in$ D2 such that x$\otimes$y where $\otimes$ represent some constraint, then if x is swapped with x', y can also be swapped with y' as long as x'$\otimes$y'. Stated another way, we can say that a child birth date in any of the databases cannot be greater than a parent birth date.

The third type of data mangling relates to the mangling of computed fields and partial computed fields. These two types of fields are considered dependent and are derived from some other fields. For example, a full name field can be a computed field as it is derived from a first name and last name. When iSTDE mangles the first name and last name, it also re-computes the full name to maintain names consistency. Partial computed fields are those fields that have partial independent values and partial computed values. For example, a contact name field can contain a brother name. It might be possible that two brothers have the same last name, so if we mangle the last name, we also need to re-compute the partial value in the contact name field.

4.5 Transferring Mangled Data

Once mangling of data is complete, the fifth step in the entire process is the automatic transfer of the de-identified test data to the user-specified unprotected environment. To do this, iSTDE creates a dump of all the temporary databases, transfers them via a secure copy to the unprotected environment, and executes remote commands to restore those dumps in databases in the unprotected environment. A significant challenge while transferring the test data was to manage the access controls and firewalls. iSTDE uses a number of built-in scripts in confidential environment to manage these network transfer obstacles.

4.6 Destroying Mappings

In this semantic-based extraction process, iSTDE produces and uses data files and domain dictionaries. The data files are created in the third step of iSTDE execution and contain extracted records from different database managers, whereas domain dictionaries are developed in the fourth step. These domain dictionaries are a sort of mapping table that helps in shuffling the records. Ideally, this step destroys all traces i.e., data files and domain dictionaries that could indentify or even hint at any sensitive information about the patients. Thus, iSTDE ensures the sensitivity of patients records by deleting the temporary databases as well as the data files mentioned above.

5 EXPERIENCES WITH ISTDE

We have used iSTDE to test several key CHARM components. For example, it was particularly useful in testing the SyncEngine because its primary function is to correlate data among multiple PP. Previously, testing the SyncEngine required developers to create test data manually. Not only was this very time consuming, but it resulted in test-data sets with limited coverage. The CHARM developers found data set produced by iSDTE to be good approximation of the real data and that they properly hid personal identities. Most importantly, the CHARM developers found several critical software faults using the iSDTE test data that would have been difficult to find with handcrafted test data. The CHARM developer had similar experiences testing the CHARM matcher and web interface.

6 FUTURE WORK AND SUMMARY

iSDTE is a work in progress and would benefit from a few enhancements. First, iSTDE currently is a desktop-based application that runs in a confidential environment by external actor. With some minor modifications it could be converted into a web-service-based application that can make it accessible without physical interaction. Second, it could be enhanced to allow incremental construction of test sets. This would allow the developers to extract an initial slice of test data and then add to it later, if a large test-data set is needed. Third, iSTDE could be further optimized to reduce execution time by better restructuring some of SQL queries that extract data from PPs and by improving the efficiency of the data mangling. Finally, the current iSTDE is tightly coupled to source databases, so addition of a new database requires developers to add modules that are specific to the new database. With some refactoring and application of the Adapter Pattern [11], this undesirable coupling could be reduced or eliminated.

In summary, preliminary indications are that iSTDE is very beneficial for testing integrated systems and merits further enhancement and study.

REFERENCES

GenerateData.com, *a tool for generating test data*, last accessed on December 22, 2008.

SqlEdit, *www.sqledit.com*, last accessed on November 28, 2008.

www.forsql.com, *a tool for automatics generating test data for developers*, last accessed on December 30, 2008.

http://www.tethyssolutions.com/T10.htm, a tool for *generating meaningful randomized data for QA testing*, load testing, last accessed on December 30, 2008.

www.tns-soft.com, *a tool for generating meaningful randomized data for QA testing*, load testing, last accessed on December 30, 2008.

CHARM Exective summary reference.

www.datanamic.com, *generate the test data from a variety of sources, including the database tables*, last accessed on December 30, 2008.

www.turbodata.ca, *generate the test data from a real database*, last accessed on February 11, 2008.

R.Maddy, *DBTESTGEN – MS Thesis 2006*, Computer Science Department, Utah State University.

Clyde, S., and Salkowitz, S., *The Unique Records Portfolio*, Public Health Informatics Institute, Decatur, GA, April, 2006.

Refactoring: Improving the Design of Existing Code (Addison-Wesley Series) by M. Fowler

http://www.sqlmanager.net/en/products/postgresql/datage nerator, *generate the test data by a variety of sources*, last accessed on February 11, 2008.

Bersano, T., Clement, B., and Shilkrot, L. Synthetic Data for Testing in Databases, University of Michigan, 1997.

Harper, K.E. *Syntactic and semantic problems in automatic sentence generation*. In Proceedings of International Conference on Computational Linguistics, 1967, 1-9.

Chays, D., Dan, S., Vokolos, F.I., and Weyuker, E.J. A *framework for testing database applications*. In ISSTA
2000, ACM 1-58113-266-2.

Phyllis F., D. Chays, *Test data generation for relational database applications*, 2004, ACM AA13115007.