# The Application of a Software Testing Technique to Uncover Data Errors in a Database System

Shirley A. Becker
&
Anthony Berkemeyer
Software Engineering & Computer Science
Florida Institute of Technology
150 West University Blvd.
Melbourne, Florida 32901
Ph: (407)674-8149 Fax: (407)674-8192
becker@cs.fit.edu

**ABSTRACT**

It is proposed that database systems be assessed for quality using an approach based on stochastic software testing. This approach requires an understanding of potential data errors ranging from syntactic mistakes to undetectable dirty data. A data error classification schema has been developed to categorize the type of data errors that could be found during testing. This classification schema provides a basis for developing test cases inclusive of data, constraints, rules, and supporting queries. A process is described whereby a database system is tested using the test cases and data validation requirements to generate feedback on data quality.

**Key Words**: Database Systems, Data Quality, Software Testing

**Biography:**

Dr. Shirley A. Becker is an associate professor of computer science at Florida Tech and co-director of its software engineering research center. Dr. Becker has published numerous articles in software engineering, team processes, and database technology. Dr. Becker is also the editor-in-charge of the Journal of Database Management.

**Published in the Pacific Northwest Software Quality Conference (PNSQC), Portland, Oregon, October 12-13, 1999.** 

# The Application of a Software Testing Technique to Uncover Data Errors in a Database System

## 1. Introduction

For many of today's operational software systems, the quality of the data is of great concern. Much of the data "mess" is hidden from end users by the filtering that occurs when a report is generated (Gordon, 1996). Unfortunately, the errors are obscured in the report that then is erroneously assumed to be correct. This false assumption means that inconsistent, missing, dirty, and other data problems virtually go undetected.

To compound the problem, many legacy systems are used without any mechanisms for identifying problems associated with data representation, standards, normalization, and redundancy, among others. Most practitioners would agree that these systems provide a wealth of historical information if the data could be relied upon as having a relatively high quality. The proliferation of data warehousing and related research to find efficient ways of relying upon historical data supports this matter (Inmon & Hackathorn, 1994). Though manual techniques have been developed to "clean" the data, there is still the issue of the prohibitive cost of using such techniques.

The underlying issue is an organization's ability to assess data quality given the size and complexity of its operational and legacy software systems (Hoxmeier, 1997). The multiplicity of data correctness problems associated with the use of software systems makes it virtually impossible to identify and fix each problem. What is needed is an effective means of obtaining feedback on the quality of the data in order to make decisions on quality improvements.

It is proposed that data be assessed for quality much the same way that software applications are assessed. There is much to be gained by applying the techniques used in software development in dealing with system size and complexity issues. Traditional testing techniques, for example, are seldom used to test all possible components of a system as was done a decade ago. It has been found that this type of coverage-based testing is infeasible given today's systems; and as a result, more sophisticated testing techniques have been developed.

One such technique, developed by Whittaker (1997), relies on stochastic testing to provide valuable feedback on a software system's defects. This testing technique is based on sampling as a means of quantifying the validity of the software system. This paper describes initial work on the application of this approach to obtain feedback on the quality of data.

The paper is organized as follows. Section 2 describes the various data quality issues that may be addressed in a testing environment. Section 3 presents the background on the proposed testing environment. Section 4 describes the testing process that would be used to uncover data errors. Section 5 concludes the paper and addresses future research opportunities.

## 2. Data Quality

The phrase "data quality" can be interpreted in many different ways depending on the organization's data requirements (Orr, 1998). Data quality may range from obvious syntactic mistakes to undetectable dirty data. It is important to understand the types of data errors that could occur in order to develop effective ways of identifying them during the testing process.

A data quality classification schema has been developed that is comprised of *correctness, completeness, comprehension, and consistency* classes (Greenfield, 1996; Fox et al., 1994). Table 1, organized by the classification schema, lists the data errors that potentially could cause major problems during the use of a software system. Of course not all of these data issues may be relevant during system use. However, it is important to realize that there are many types of data errors that could occur with or without actually causing a system failure. Each of the data quality classes is briefly described below.

- *Data consistency* is concerned with ensuring that data is represented "semantically the same way" within and across tables in a database system. Data consistency is not maintained when data with the same meaning physically have varying data types and lengths. Data inconsistencies may occur when data elements are hidden in aggregated data. Data consistency is also impacted when integrity constraints are not used properly. Inconsistencies may occur when child records are not associated with a parent record or when key values don't match due to nulls or inaccuracies (e.g., spelling mistakes) in data elements.

- *Data completeness* is associated with missing or inaccessible data as a result of inadvertently storing nulls, blanks, abbreviations, truncations, or partial data. Completeness problems become an issue when integrity constraints are missing or inappropriately enforced as in the example of cascading updates or deletions across tables.

- *Data correctness* is concerned with corrupt or wrong data being stored; as well as, dirty data being shown to the user under the guise of valid data. Data corruption occurs, for example, when data is incorrectly manipulated through the use of views. Nonexistent data is shown to the user when incorrect joins are performed over nonkey attributes (refer to Date(1995) for a discussion of loss projection). Though this doesn't impact data storage, it can cause major problems for the user when the data is assumed to be valid. Data correctness is also impacted when integrity constraints are incorrectly used or inadvertently disabled. Check, unique, or null constraints may be too restrictive causing problems with data insertion, updates, and deletions. Finally, data entry errors cause problems when misspelled, missing, partial, or wrong data is stored.

- *Data comprehension* problems may be inherited from legacy systems that contain cryptic, obsolete or unknown data due to changes in the real world applications (e.g., zip code change from 5 to 9 digits). When data is aggregated or concatenated instead of being stored as a set of attributes, vital details may be lost.

**Table 1: Data Quality Factors**

| Factor: | Description: | Example: |
|---|---|---|
| *Data Consistency Issues*: | | |
| Varying Data Definitions | The data type and length for a particular attribute may vary in tables though the semantic definition is the same. | Social security number may be defined as: Number (9) in one table and Varchar2(11) in another table. |
| Varying Data Codes & Values | The data representation of the same attribute may vary within and across tables. | A flag representing yes or no may be defined in many ways. The value 'no' may be represented as: 'NO', 'N','n','0','1', or blank. |
| Misuse of Integrity Constraints | When referential integrity constraints are misused, foreign key values may be left "dangling" or inadvertently deleted. | An employee record is deleted but his/her dependent records are not deleted. |
| Nulls | Nulls may be ignored when joining tables or doing searches on the column. | The supervisor (Smith) has been entered as a null value for an employee (Jones). A report of all employees supervised by Smith would not list Jones. |
| *Data Completeness Issues*: | | |
| Missing data | Data elements are missing because of a lack of integrity constraints or nulls are inadvertently not updated. | A shipment's date of estimated arrival is null thus impacting an assessment of variances in estimated/actual arrival data. |
| Inaccessible Data | Inaccessible record due to missing or redundant unique identifier value. | Customer numbers are used to identify a customer record. Because uniqueness was not enforced, the customer ID (45656) identifies more than one customer. |
| Missing Integrity Constraints | Missing constraints can cause data errors due to nulls, nonuniqueness, or missing relationships. | Part records with a supplier identifier exist in the database but cannot be matched to an existing supplier. |
| *Data Correctness Issues*: | | |
| Loss Projection | Tables that are joined over nonkey attributes will produce nonexistent data that is shown to the user. | Lisa Evans works in the LA office in the Accounting department. When a report is generated, it shows her working in Marketing and Accounting. |
| Incorrect Data Values | Data that is misspelled or inaccurately recorded. | 123 Maple Street is recorded with a spelling mistake and a street abbreviation (123 Mapel St) |
| Inappropriate Use of Views | Data is updated incorrectly through views. | There is a view that contains nonkey attributes from base tables. When it is used to update the database, null values are entered into the key columns of the base tables. |
| Disabled Integrity Constraints | Null, nonunique, or out of range data may be stored when the integrity constraints are disabled. | The primary key constraint is disabled during an import function. Data is entered into the existing data with null unique identifiers. |
| Misuse of Integrity Constraints | Check, not null, or foreign key constraints are inappropriate or too restrictive. | Check constraint only allows hardcoded values of "C", "A", "X", and "Z". But a new code "B" cannot be entered. |
| *Data Comprehension Issues*: | | |
| Data Aggregation | Aggregated data is used to represent a set of data elements. | One name field is used to store surname, firstname, middle initial, and last name (e.g., John, Hanson, Mr.). |
| Cryptic Object Definitions | Database object (e.g., column) has a cryptic, unidentifiable name. | Customer table with a column labeled, "c_avd". There is no documentation as to what the column might contain. |
| Unknown or Cryptic Data | Cryptic data stored as codes, abbreviations, truncated, or with no apparent meaning. | Shipping codes used to represent various parts of the customer base ('01', '02', '03'). No supporting document to explain the meaning of the codes. |

What is needed is a means of identifying these data errors in order improve and maintain the quality of a software system. We propose the use of a software testing technique to assist in uncovering data errors while validating system behavior.

## 3. Testing

The sampling used for stochastic testing of a software system is an important aspect of this work because of its representation of the real world population. With a sufficiently large and unbiased sample, feedback can be obtained not only on the software reliability but also on the quality of the data. Expensive, exhaustive testing techniques can be replaced with sampling to identify problem areas in metadata (database objects) and data.

There are several advantages to using this approach in assessing data quality. There is a certain statistical confidence in the test results based on sample size. As the sample size increases so does the confidence in the test results as being representative of the whole population (refer to Whittaker & Poore (1993) for an in-depth discussion). In addition, those components of the system that are used most often or are of greater interest than other components would appear in a significant number of test cases as testing is based on usage.

The testing process that provides a basis for this work is presented in Table 2. This approach requires a profile of the quality factors to be included as part of the testing activity. The profile of the quality factors includes validation criteria and data requirements necessary to determine the success or failure of the system once a test case has been executed. The table identifies the artifacts that would be produced after each process step. The final step produces a set of logs that would include the number of system defects, an audit log of data changes, and a data error log that would contain data errors associated with each quality factor under assessment.

**Table 2: General Description of the Testing Process**

| Process Step | Explanation | Artifacts |
|---|---|---|
| Identify quality factor(s) to be assessed. | The quality factors drive the data validation criteria necessary for effective testing. | Quality factors. |
| Model the system component. | The set of states representing the component's behavior is constructed in a directed graph. Each arc is assigned a probability between 0 and 1 (inclusive). The sum of the exit arcs probabilities from each state = 1. | Behavior model. |
| Determine sample size. | Sample size is based on statistical confidence and reliability. | Sample size. |
| Develop test sequences. | Test sequences are generated based on usage probabilities. | Test sequences. |
| Develop test data & data validation criteria. | Test data is generated for each sequence inclusive of quality factors & data validation criteria. | Test cases with validation criteria. |
| Perform testing and capture results. | The test cases are executed & defects/data errors logged. Data may be gathered before and after test case execution based on validation requirements. | Audit log, data error report, & defect log. |

Figure 1 depicts the testing environment whereby the inputs, process, and outputs provide the basis for obtaining feedback on the quality of the data. The inputs of this testing environment would include a sample of test cases with database transactions of reads, updates, deletions, and insertions. Inputs also include the selected data quality factors with associated validation criteria, which become the basis for analyzing the test results.

The behavior model, shown in the middle of the figure, represents the system's states and usage probabilities. The usage probability associated with each arc is based on past experience, extracted from audit logs, or randomly generated.

The outputs produced by the testing activities would include the number of defects (as defined in the traditional sense of testing), database audit logs that would contain old and new data values, and a data error report. The output requirements are determined by the validation criteria as specified as part of the inputs of the testing environment. Based on the validation criteria, the audit log may be updated and the data error report may contain the output of SQL queries that were executed to provide additional information about the quality of the data.

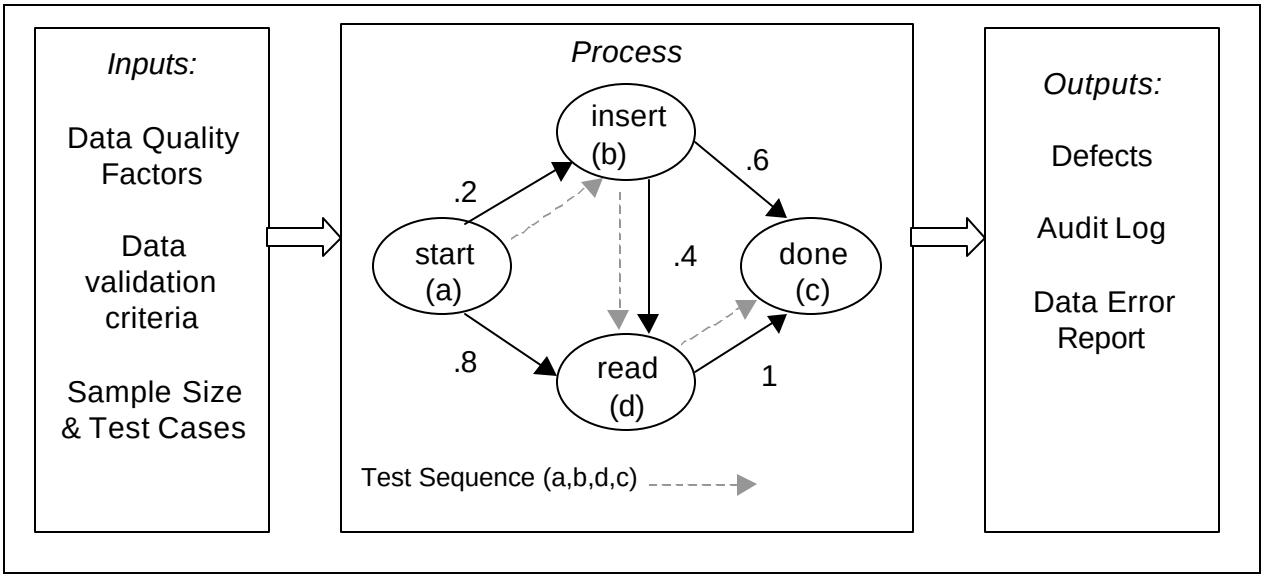


*Figure 1: The Testing Environment*

The next section will provide an overview of the testing environment in terms of a simple application for updating projects.

## 4. The Project Update Application

A behavior model (Figure 2a) represents a project update application (Figure 2b), which is part of a web-based project management tool (Becker & Ladino, 1999). The arcs in the model are labeled with probabilities of use to represent the real-world activities associated with project updates. The model shows that ninety percent of all project updates are successful in that project records were found and data modified. This is reflected as the commit state that ensures changes are made permanently to the database system. The rollback state, which has a ten-percent

probability of occurrence, is executed when the update fails thus ensuring that any temporary changes are not permanently recorded in the database system.

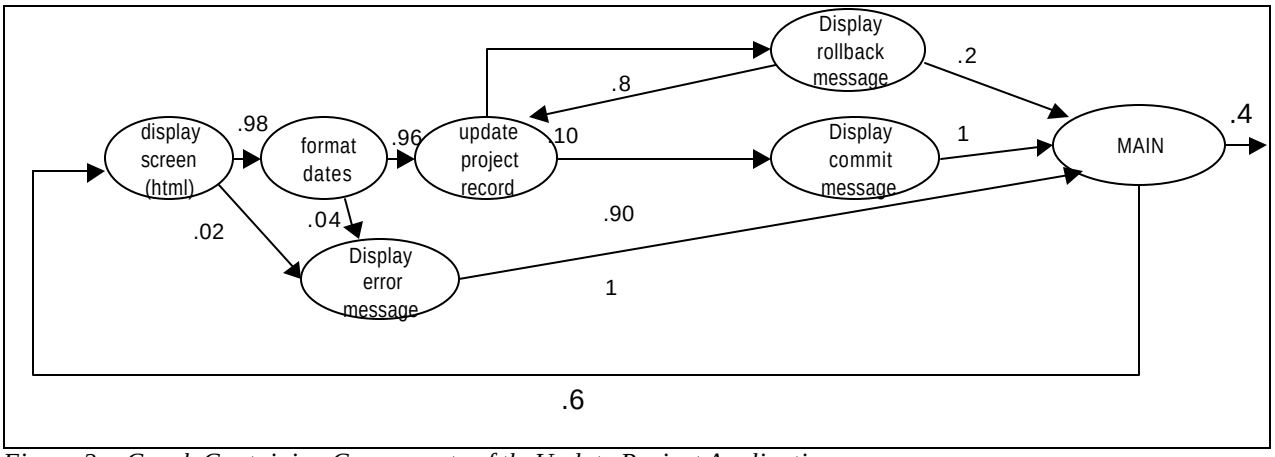


*Figure 2a: Graph Containing Components of theUpdate Project Application*

```
/************************************************
*  This procedure updates the table PROJECTS   *
************************************************/
PROCEDURE u_projects
   (p_id              IN projects.id%TYPE,
    p_name            IN projects.name%TYPE,
    p_manager         IN projects.manager%TYPE,
    p_sdate           IN owa_util.dateType,
    p_edate           IN owa_util.dateType)
  IS
    pr_edate DATE;
    pr_sdate DATE;

/*web-based application */
  BEGIN
   htp.htmlOpen;
   htp.bodyOpen;
   pr_sdate := owa_util.todate(p_sdate);
   pr_edate := owa_util.todate(p_edate);

/*  SQL component */
    UPDATE projects
     SET name      = p_name,
        mgr_name = p_manager,
        start_date = pr_sdate,
        end_date   = pr_edate
     WHERE id = p_id;
   COMMIT;

/* Error Component for the Rollback */
…
END u_projects;
```

*Figure 2b: Part of the Code Design for Update Project Application (Becker & Ladino, 1999)*

This example exhibits such simple behavior that one would think there was little opportunity for data problems to occur as the unique identifier either matches a record or is not found in the database. Even in this simple example, however, there is the potential for a range of data errors. These data errors would not typically be found during traditional means of software testing, as a defect would constitute a system failure. System failure, for example, could occur during the execution of the html statements, date function calls, or the SQL. The testing results would uncover defects in the execution of the code but not necessarily data corruption or integrity problems. To illustrate this point, Table 3 identifies potential data problems associated with the update project application that would not have resulted in a system defect.

**Table 3: Potential Transaction and Data Errors That May Go Undetected.**

| *Transaction Problem* |
|---|
| The SQL syntax doesn't deal with upper or lower case data resulting in "no record found" when the record should have been found (e.g., 'P123' <> 'p123'). |
| *Data Problems* |
| The record identifier is not unique. This could be the result of a missing integrity constraint (unique or primary key) or syntax (both 'p123' and 'P123' are in the database as separate records). |
| The manager's name is a long text field. This may cause data problems in terms of the ordering of the surname and first and last names ('Jones, Sally, Ms.'or 'Ms. Sally Jones'), missing data elements (e.g., no surname is included), and syntactic (spelling) mistakes. |
| The manager's id must match an existing employee social security number to maintain referential integrity. A missing integrity constraint would not ensure this matching occurs thus allowing a manager to exist without a matching employee record. |

In order to follow the testing process that has been proposed, each test case must include both data and data validation criteria in order to identify system defects and data errors. Test cases are selected based on the probabilities of use as represented in the behavior model.

We start by identifying test sequences that reflect how the system is used. For our example, test sequences are illustrated by the following:

Sequence 1: *display screen(A), get today's date(B), update project(C), commit(D), quit(F).*
Sequence 2: *display screen(A), get today's date(B), update project(C), rollback(E), update project(C), commit(D), quit(F).*
Sequence 3: *display screen(A), get today's date(B), update project(C), rollback(E), quit(F).*

Next, the data validation criteria are added based on selected data quality factor(s). The quality factor under investigation is *missing integrity constraints* as part of a completeness check. The data validation criteria will include both primary key and foreign key checks to ensure that as projects are updated data integrity remains intact. In this case, the foreign key constraint is on manager's id (social security number), which means that a manager cannot be assigned to a project unless he or she exists as an employee in the Employee table.

The data validation criteria are expressed as a set of rules that would be used to determine whether there are data errors resulting from the execution of the test case. Table 4 shows several

data validation criteria associated with the first test sequence (A,B,C,D,F). These rules and their links to data quality factors would reside as physical data in the database.

**Table 4: Illustration of Data Validation Criteria for Primary Key Integrity Constraint**

| Rule: | Validation |
|---|---|
| 1. One and only one record found by project_id | One record returned. If more than one record, then update error report. A query is executed to check database objects for disabled primary key constraint. |
| 2. When no records found, execute "like" query. | No records were found. A query is executed to search for similar text patterns matched against the p_id. Error report is updated with similar p_ids. |
| 3. When no records found, execute query to search for missing values (nulls). | No records were found. A query is executed to determine if there are missing records identifiers. Error report is updated with existence or nonexistence of nulls. |

Depending on the rule, we may need to track old and new data values in order to determine whether a data error has occurred. This requirement is also stored with the rule and becomes part of the information associated with each test sequence. When the rule specifies that old and new data values need to be stored, an audit log entry is made during testing activities. In our example, the primary key integrity validation would not require an audit log entry because we are only concerned with finding one record matching on the primary key. For other quality factors, however, it would be imperative to have old and new data values to ensure that data errors have not occurred.

Table 5 shows a test case whereby data and validation criteria data have been added to test sequence (A,B,C,D,F). As part of the data validation criteria, the test case identifies potential queries that would be used to study data errors after test case execution. In this case, SQL queries would produce additional information used to assess whether the integrity constraints are enabled, and to determine whether the project identifier exists in the database. For manager's social security number, the Employee table is searched for data type and data values similar to the test case data.

Once the testing activity is completed, the data errors need to be analyzed in terms of the overall impact on the database system. The type and number of data errors discovered may require further study to determine the best approach to resolving them. In the test case presented above, integrity constraints may be enabled to ensure data integrity for future use. More importantly, would be the need for data cleaning or conversion depending on the severity of the problems found. Though this discussion is beyond the scope of this paper, future research efforts need to address automated tools for data maintenance.

## 5. Conclusion and Future Research

It is proposed in this paper that stochastic testing techniques be applied to the testing of data quality in software systems. This type of testing allows for a sampling of software behavior that would represent real world use of the system. In this respect, the software applications that are used most often would have a higher representation in the test cases generated.

**Table 5: Test Case (A,B,C,D,F) and Potential Data Errors**

| Test Data | Potential Data Errors due to lack of Integrity Constraints |
|---|---|
| DATA:<br>'P123'<br>'Audit Features'<br>'344-78-9856'<br>'02-10-1999'<br>'09-01-1999'<br><br>QUALITY FACTOR:<br>integrity constraints<br><br>RULES:<br>pk: one record with correct id<br>fk: manager identifier is in the Employee table | PRIMARY KEY:<br>• 'P123' does not uniquely identify one project. (Note: *this would not result in a test defect if the primary key integrity constraint were disabled*).<br>• 'P123' does uniquely identify one project but is stored as 'p123'. (Note: *this would not result in a test defect, as the record would not be found.)* |
| SUPPORTING QUERIES:<br>Primary Key:<br>select * from Project where p_id like '&123';<br>select count(*) from Project where p_id is null;<br>Foreign Key:<br>select * from Employee where emp_id like '344&';<br>Both: /*check for enabled constraints*/<br>select * from user_constraints; | FOREIGN KEY:<br>• The manager (SS#344-78-9856) does not exist in the Employee table. (Note: *this would not result in a test defect but would cause a data error*).<br>• The manager (SS#344-78-9856) is in the Employee table but the data is stored as a number data type (344789856). (Note: *this would not result in a test defect but would cause a data error)*. |

Traditional testing of software systems identifies defects in terms of the successful execution of a particular test case. The problem with this approach when searching for data errors is that the execution path may be completed successfully while data errors go undetected. What is needed is a means of testing software applications in terms of defects and data errors.

A testing environment has been described that would include the selection of data quality factors as part of the testing activities. Each data quality factor would have a set of rules that would identify validation criteria and expected outcomes. The validation criteria would provide the means for assessing data quality associated with each test case. As a result, testing would include an evaluation of data appropriate to the quality factor that has been selected.

It is important to develop automated means (e.g., SQL queries) that would supplement each test case execution. An automated environment is currently being studied to support the rule-based validation criteria. Several operational database systems have been analyzed using this approach in order to initiate the development of a common set of validation criteria rules associated with quality factors.

There is a significant amount of learning potential as the result of data analysis and error reporting. Further study is needed to ensure that the validation criteria rules and supporting queries are updated based on testing experience. These findings need to be incorporated into the data and data objects that represent the quality factors and supporting rule set.

Finally, a severity rating system needs to be developed that would provide feedback on the impact of the data errors encountered. It may be the case that the data error is trivial and doesn't impact the use of the system. But, in most cases, the data errors will point to a much larger problem in that data is incorrect, inaccessible, missing, or corrupted.

**References**


Becker, S. and Ladino, D. (1999). "A Technical Infrastructure for Process Support," **Software Process Improvement: Concepts and Practices,** IDEA Group Publishing, Hershey, PA.

Date, C.J. (1995). **Introduction to Database Systems,** Addison_Wesley Publishing, Reading, MA.

Gordon, K.I. (1996). "The Why of Data Standards – Do You Really Know Your Data," http://www.island.net/~gordon/whystds.htm.

Greenfield, L. (1997). "An (informal) Taxonomy of Data Warehouse Data Errors," http:/pwp.starnetinc.com/larryg/errors.html.

Fox, C., Levitin, A. & Redman, T. (1994). "The Notion of Data and Its Quality Dimensions," **Information Processing and Management,** Vol. 30, No. 1.

Hoxmeier, J. (1997). "A Framework for Assessing Database Management**," Proceedings of the ER'97 Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling,** Los Angeles, CA.

Inmon, W. H. and Hackathorn, R.D. (1994). **Using the Data Warehouse.** Wiley & Sons NY, NY.

Orr, K. (1998). "Data Quality and Systems Theory," **Communications of the ACM**, Vol. 41, No. 2.

Whittaker, J. (1996). "Certification Practices," **Cleanroom Software Engineering Practices,** IDEA Group Publishing, Hershey, PA.

Whittaker, J. (1997). "Stochastic Software Testing," **Annals of Software Engineering**, Vol. 4.

Whittaker, J. & Poore, J. (1993). "Markov Analysis of Software Specifications," **ACM Transactions on Software Engineering and Methodology**, Vol.2, No.1.