A Comprehensive Approach to Data Warehouse Testing

Matteo Golfarelli
DEIS - University of Bologna
Via Sacchi, 3
Cesena, Italy
matteo.golfarelli@unibo.it

Stefano Rizzi
DEIS - University of Bologna
VIale Risorgimento, 2
Bologna, Italy
stefano.rizzi@unibo.it

ABSTRACT

Testing is an essential part of the design life-cycle of any software product. Nevertheless, while most phases of data warehouse design have received considerable attention in the literature, not much has been said about data warehouse testing. In this paper we introduce a number of data mart-specific testing activities, we classify them in terms of what is tested and how it is tested, and we discuss how they can be framed within a reference design methodology.

Categories and Subject Descriptors

H.4.2 [**Information Systems Applications**]: Types of Systems—*Decision support*; D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Verification, Design

Keywords

data warehouse, testing

1. INTRODUCTION

Testing is an essential part of the design life-cycle of any software product. Needless to say, testing is especially critical to success in data warehousing projects because users need to trust in the quality of the information they access. Nevertheless, while most phases of data warehouse design have received considerable attention in the literature, not much has been said about data warehouse testing.

As agreed by most authors, the difference between testing data warehouse systems and generic software systems or even transactional systems depends on several aspects [21, 23, 13]:

- Software testing is predominantly focused on program code, while data warehouse testing is directed at data and information. As a matter of fact, the key to data warehouse testing is to know the data and what the answers to user queries are supposed to be.
- Differently from generic software systems, data warehouse testing involves a huge data volume, which significantly impacts performance and productivity.
- Data warehouse testing has a broader scope than software testing because it focuses on the correctness and usefulness of the information delivered to users. In fact, data validation is one of the main goals of data warehouse testing.
- Though a generic software system may have a large number of different use scenarios, the valid combinations of those scenarios are limited. On the other hand, data warehouse systems are aimed at supporting any views of data, so the possible combinations are virtually unlimited and cannot be fully tested.
- While most testing activities are carried out before deployment in generic software systems, data warehouse testing activities still go on after system release.
- Typical software development projects are self-contained. Data warehousing projects never really come to an end; it is very difficult to anticipate future requirements for the decision-making process, so only a few requirements can be stated from the beginning. Besides, it is almost impossible to predict all the possible types of errors that will be encountered in real operational data. For this reason, regression testing is inherently involved.

Like for most generic software systems, different types of tests can be devised for data warehouse systems. For instance, it is very useful to distinguish between *unit test*, a white-box test performed on each individual component considered in isolation from the others, and *integration test*, a black-box test where the system is tested in its entirety. Also *regression test*, that checks that the system still functions correctly after a change has occurred, is considered to be very important for data warehouse systems because of their ever-evolving nature. However, the peculiar characteristics of data warehouse testing and the complexity of data warehouse projects ask for a deep revision and contextualization of these test types, aimed in particular at emphasizing the relationships between testing activities on the one side, design phases and project documentation on the other.

From the methodological point of view we mention that, while testing issues are often considered only during the very

DOLAP'09, November 6, 2009, Hong Kong, China.

last phases of data warehouse projects, all authors agree that advancing an accurate test planning to the early projects phases is one of the keys to success. The main reason for this is that, as software engineers know very well, the earlier an error is detected in the software design cycle, the cheapest correcting that error is. Besides, planning early testing activities to be carried out during design and before implementation gives project managers an effective way to regularly measure and document the project progress state.

Since the correctness of a system can only be measured with reference to a set of requirements, a successful testing begins with the gathering and documentation of end-user requirements [8]. Since most end-users requirements are about data analysis and data quality, it is inevitable that data warehouse testing primarily focuses on the ETL process on the one hand (this is sometimes called *back-end testing* [23]), on reporting and OLAP on the other (*front-end testing* [23]). While back-end testing aims at ensuring that data loaded into the data warehouse are consistent with the source data, front-end testing aims at verifying that data are correctly navigated and aggregated in the available reports.

From the organizational point of view, several roles are involved with testing [8]. Analysts draw conceptual schemata, that represent the users requirements to be used as a reference for testing. Designers are responsible for logical schemata of data repositories and for data staging flows, that should be tested for efficiency and robustness. Testers develop and execute test plans and scripts. Developers perform white box unit tests. Database administrators test for performance and stress, and set up test environments. Finally, end-users perform functional tests on reporting and OLAP front-ends.

In this paper we propose a comprehensive approach to testing data warehouse systems. More precisely, considering that data warehouse systems are commonly built in a bottom-up fashion, by iteratively designing and implementing one data mart at a time, we will focus on the test of a single data mart. The main features of our approach can be summarized as follows:

- A consistent portion of the testing effort is advanced to the design phase to reduce the impact of error correction.
- A number of data mart-specific testing activities are identified and classified in terms of *what* is tested and *how* it is tested.
- A tight relationship is established between testing activities and design phases within the framework of a reference methodological approach to design.
- When possible, testing activities are related to quality metrics to allow their quantitative assessment.

The remainder of the paper is organized as follows. After briefly reviewing the related literature in Section 2, in Section 3 we propose the reference methodological framework. Then, Section 4 classifies and describes the data mart-specific testing activities we devised, while Section 5 briefly discusses some issues related to test coverage. Section 6 proposes a timeline for testing in the framework of the reference design methodology, and Section 7 summarizes the lessons we learnt.

2. RELATED WORKS

The literature on software engineering is huge, and it includes a detailed discussion of different approaches to the testing of software systems (e.g., see [16, 19]). However, only a few works discuss the issues raised by testing in the data warehousing context.

[13] summarizes the main challenges of data warehouse and ETL testing, and discusses their phases and goals distinguishing between *retrospective* and *prospective* testing. It also proposes to base part of the testing activities (those related to incremental load) on mock data.

In [20] the authors propose a basic set of attributes for a data warehouse test scenario based on the IEEE 829 standard. Among these attributes, the test purpose, its performance requirements, its acceptance criteria, and the activities for its completion.

[2] proposes a process for data warehouse testing centered on a *unit test* phase, an *integration test* phase, and a *user acceptance test* phase.

[21] reports eight classical mistakes in data warehouse testing; among these: not closely involving end users, testing reports rather than data, skipping the comparison between data warehouse data and data source data, and using only mock data. Besides, unit testing, system testing, acceptance testing, and performance testing are proposed as the main testing steps.

[23] explains the differences between testing OLTP and OLAP systems and proposes a detailed list of testing categories. It also enumerates possible test scenarios and different types of data to be used for testing.

[8] discusses the main aspects in data warehouse testing. In particular, it distinguishes the different roles required in the testing team, and the different types of testing each role should carry out.

[4] presents some lessons learnt on data warehouse testing, emphasizing the role played by constraint testing, source-to-target testing, and validation of error processing procedures.

All papers mentioned above provide useful hints and list some key testing activities. On the other hand, our paper is the first attempt to define a comprehensive framework for data mart testing.

3. METHODOLOGICAL FRAMEWORK

To discuss how testing relates to the different phases of data mart design, we adopt as a methodological framework the one described in [7]. As sketched in Figure 1, this framework includes eight phases:

- *Requirement analysis*: requirements are elicited from users and represented either informally by means of proper glossaries or formally (e.g., by means of goal-oriented diagrams as in [5]);
- *Analysis and reconciliation*: data sources are inspected, normalized, and integrated to obtain a reconciled schema;
- *Conceptual design*: a conceptual schema for the data mart –e.g., in the form of a set of fact schemata [6]– is designed considering both user requirements and data available in the reconciled schema;
- *Workload refinement*: the preliminary workload expressed by users is refined and user profiles are singled out, using for instance UML use case diagrams;

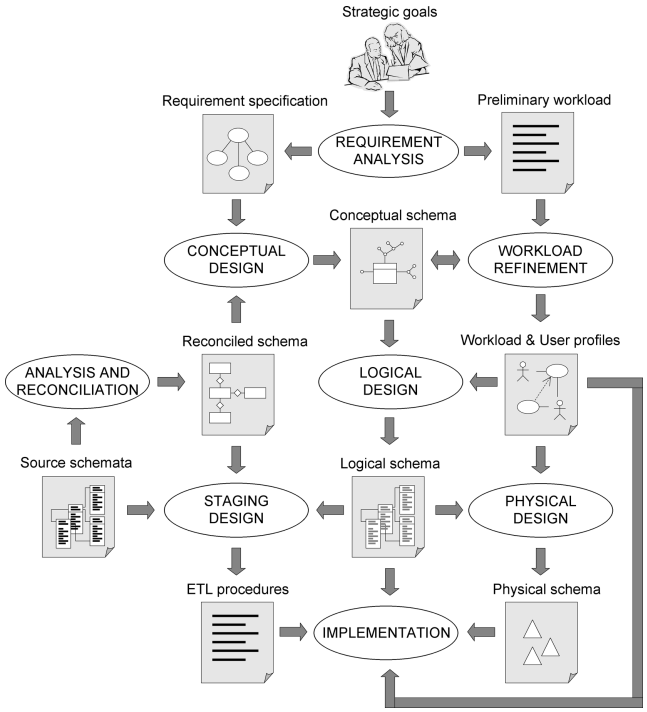

Figure 1: Reference design methodology

- *Logical design*: a logical schema for the data mart (e.g., in the form of a set of star schemata) is obtained by properly translating the conceptual schema;
- *Data staging design*: ETL procedures are designed considering the source schemata, the reconciled schema, and the data mart logical schema; and
- *Physical design*: this includes index selection, schema fragmentation, and all other issues related to physical allocation.
- *Implementation*: this includes implementation of ETL procedures and creation of front-end reports.

Note that this methodological framework is general enough to host both supply-driven, demand-driven, and mixed approaches to design. While in a supply-driven approach conceptual design is mainly based on an analysis of the available source schemata [6], in a demand-driven approach user requirements are the driving force of conceptual design [3]. Finally, in a mixed approach requirement analysis and source schema analysis are carried out in parallel, and requirements are used to actively reduce the complexity of source schema analysis [1].

4. TESTING ACTIVITIES

In order to better frame the different testing activities, we identify two distinct, though not independent, classification coordinates: *what* is tested and *how* it is tested.

As concerns the first coordinate, *what*, we already mentioned that testing data quality is undoubtedly at the core of data warehouse testing. Testing data quality mainly entails an accurate check on the correctness of the data loaded by ETL procedures and accessed by front-end tools. However, in the light of the complexity of data warehouse projects and of the close relationship between good design and good performance, we argue that testing the *design* quality is almost equally important. Testing design quality mainly implies verifying that user requirements are well expressed by the conceptual schema of the data mart and that the conceptual and logical schemata are well-built. Overall, the items to be tested can then be summarized as follows:

- *Conceptual schema*: it describes the data mart from an implementation-independent point of view, specifying the facts to be monitored, their measures, the hierarchies to be used for aggregation. Our methodological framework adopts the Dimensional Fact Model to this end [6].
- *Logical schema*: it describes the structure of the data repository at the core of the data mart. If the implementation target is a ROLAP platform, the logical schema is actually a relational schema (typically, a star schema or one of its variants).
- *ETL procedures*: the complex procedures that are in charge of feeding the data repository starting from data sources.
- *Database*: the repository storing data.
- *Front-end*: the applications accessed by end-users to analyze data; typically, these are either static reporting tools or more flexible OLAP tools.

As concerns the second coordinate, *how*, the eight types of test that, in our experience, best fit the characteristics of data warehouse systems are summarized below:

- *Functional test*: it verifies that the item is compliant with its specified business requirements.
- *Usability test*: it evaluates the item by letting users interact with it, in order to verify that the item is easy to use and comprehensible.
- *Performance test*: it checks that the item performance is satisfactory under typical workload conditions.
- *Stress test*: it shows how well the item performs with peak loads of data and very heavy workloads.
- *Recovery test*: it checks how well an item is able to recover from crashes, hardware failures and other similar problems.
- *Security test*: it checks that the item protects data and maintains functionality as intended.
- *Regression test*: It checks that the item still functions correctly after a change has occurred.

Remarkably, these types of test are tightly related to six of the software quality factors described in [12]: correctness, usability, efficiency, reliability, integrity, flexibility.

The relationship between what and how is summarized in Table 1, where each check mark points out that a given type of test should be applied to a given item. Starting from this table, in the following subsections we discuss the main testing activities and how they are related to the methodological framework outlined in Section 3.

Table 1: What vs. how in testing

	Conceptual schema	Logical schema	ETL procedures	Database	Front-end
Functional	✓	✓	✓		✓
Usability	✓	✓			✓
Performance		✓	✓	✓	✓
Stress			✓	✓	✓
Recovery			✓	✓	
Security			✓	✓	✓
Regression	✓	✓	✓	✓	✓
	Analysis & design		Implementation		

We preliminarily remark that a requirement for an effective test is the early definition, for each testing activity, of the necessary conditions for passing the test. These conditions should be verifiable and quantifiable. This means that proper metrics should be introduced, together with their acceptance thresholds, so as to get rid of subjectivity and ambiguity issues. Using quantifiable metrics is also necessary for automating testing activities.

While for some types of test, such as performance tests, the metrics devised for generic software systems (such as the maximum query response time) can be reused and acceptance thresholds can be set intuitively, for other types of test data warehouse-specific metrics are needed. Unfortunately, very few data warehouse-specific metrics have been defined in the literature. Besides, the criteria for setting their acceptance thresholds often depend on the specific features of the project being considered. So, in most cases, ad hoc metrics will have to be defined together their thresholds. An effective approach to this activity hinges on the following main phases [18]:

1. Identify the goal of the metrics by specifying the underlying hypotheses and the quality criteria to be measured.

2. Formally define the metrics.

3. Theoretically validate the metrics using either axiomatic approaches or measurement theory-based approaches, to get a first assessment of the metrics correctness and applicability.

4. Empirically validate the metrics by applying it to data of previous projects, in order to get a practical proof of the metrics capability of measuring their goal quality criteria. This phase is also aimed at understanding the metrics implications and fixing the acceptance thresholds.

5. Apply and accredit the metrics. The metrics definitions and thresholds may evolve in time to adapt to new projects and application domains.

4.1 Testing the Conceptual Schema

Software engineers know very well that the earlier an error is detected in the software design cycle, the cheapest correcting that error is. One of the advantages of adopting a data warehouse methodology that entails a conceptual design phase is that the conceptual schema produced can be thoroughly tested for user requirements to be effectively supported.

We propose two main types of test on the data mart conceptual schema in the scope of functional testing. The first, that we call *fact test*, verifies that the workload preliminarily expressed by users during requirement analysis is actually supported by the conceptual schema. This can be easily achieved by checking, for each workload query, that the required measures have been included in the fact schema and that the required aggregation level can be expressed as a valid grouping set on the fact schema. We call the second type of test a *conformity test*, because it is aimed at assessing how well conformed hierarchies have been designed. This test can be carried out by measuring the sparseness of the bus matrix [7], which associates each fact with its dimensions, thus pointing out the existence of conformed hierarchies. Intuitively, if the bus matrix is very sparse, the designer probably failed to recognize the semantic and structural similarities between apparently different hierarchies. Conversely, if the bus matrix is very dense, the designer probably failed to recognize the semantic and structural similarities between apparently different facts.

Other types of test that can be executed on the conceptual schema are related to its understandability, thus falling in the scope of usability testing. An example of a quantitative approach to this test is the one described in [18], where a set of metrics for measuring the quality of a conceptual schema from the point of view of its understandability are proposed and validated. Example of these metrics are the average number of levels per hierarchy and the number of measures per fact.

4.2 Testing the Logical Schema

Testing the logical schema before it is implemented and before ETL design can dramatically reduce the impact of errors due to bad logical design. An effective approach to functional testing consists in verifying that a sample of queries in the preliminary workload can correctly be formulated in SQL on the logical schema. We call this the *star test*. In putting the sample together, priority should be given to the queries involving irregular portions of hierarchies (e.g., those including many-to-many associations or cross-dimensional attributes), those based on complex aggregation schemes (e.g., queries that require measures aggregated through different operators along the different hierarchies), and those leaning on non-standard temporal scenarios (such as yesterday-for-today).

In the scope of usability testing, in [17] some simple metrics based on the number of fact tables and dimension tables in a logical schema are proposed. These metrics can be adopted to effectively capture schema understandability.

Finally, a performance test can be carried out on the logical schema by checking to what extent it is compliant with the multidimensional normal forms, that ensure summarizability and support an efficient database design [11, 10].

Besides the above-mentioned metrics, focused on usability and performance, the literature proposes other metrics

for database schemata, and relates them to abstract quality factors. For instance, in the scope of maintainability, [15] introduces a set of metrics aimed at evaluating the quality of a data mart logical schema with respect to its ability to sustain changes during an evolution process.

4.3 Testing the ETL Procedures

ETL testing is probably the most complex and critical testing phase, because it directly affects the quality of data. Since ETL is heavily code-based, most standard techniques for generic software system testing can be reused here.

A functional test of ETL is aimed at checking that ETL procedures correctly extract, clean, transform, and load data into the data mart. The best approach here is to set up unit tests and integration tests. Unit tests are white-box test that each developer carries out on the units (s)he developed. They allow for breaking down the testing complexity, and they also enable more detailed reports on the project progress to be produced. Units for ETL testing can be either vertical (one test unit for each conformed dimension, plus one test unit for each group of correlated facts) or horizontal (separate tests for static extraction, incremental extraction, cleaning, transformation, static loading, incremental loading, view update); the most effective choice mainly depends on the number of facts in the data marts, on how complex cleaning and transformation are, and on how the implementation plan was allotted to the developers. In particular, crucial aspects to be considered during the loading test are related to both dimension tables (correctness of roll-up functions, effective management of dynamic hierarchies and irregular hierarchies), fact tables (effective management of late updates), and materialized views (use of correct aggregation functions).

After unit tests have been completed, an integration test allows the correctness of data flows in ETL procedures to be checked. Different quality dimensions, such as data coherence (the respect of integrity constraints), completeness (the percentage of data found), and freshness (the age of data) should be considered. Some metrics for quantifying these quality dimensions have been proposed in [22].

During requirement analysis the designer, together with users and database administrators, should have singled out and ranked by their gravity the most common causes of faulty data, aimed at planning a proper strategy for dealing with ETL errors. Common strategies for dealing with errors of a given kind are "automatically clean faulty data", "reject faulty data", "hand faulty data to data mart administrator", etc. So, not surprisingly, a distinctive feature of ETL functional testing is that it should be carried out with at least three different databases, including respectively (i) correct and complete data, (ii) data simulating the presence of faulty data of different kinds, and (iii) real data. In particular, tests using dirty simulated data are sometimes called *forced-error tests*: they are designed to force ETL procedures into error conditions aimed at verifying that the system can deal with faulty data as planned during requirement analysis.

Performance and stress tests are complementary in assessing the efficiency of ETL procedures. Performance tests evaluate the behavior of ETL with reference to a routine workload, i.e., when a domain-typical amount of data has to be extracted and loaded; in particular, they check that the processing time be compatible with the time frames expected for data-staging processes. On the other hand, stress tests simulate an extraordinary workload due to a significantly larger amount of data.

The recovery test of ETL checks for robustness by simulating faults in one or more components and evaluating the system response. For example, you can cut off the power supply while an ETL process is in progress or you can set a database offline while an OLAP session is in progress to check for restore policies' effectiveness.

As concerns ETL security, there is a need for verifying that the database used to temporarily store the data being processed (the so-called *data staging area*) cannot be violated, and that the network infrastructure hosting the data flows that connect the data sources and the data mart is secure.

4.4 Testing the Database

We assume that the logical schema quality has already been verified during the logical schema tests, and that all issues related to data quality are in charge of ETL tests. Then, database testing is mainly aimed at checking the database performances using either standard (performance test) or heavy (stress test) workloads. Like for ETL, the size of the tested databases and their data distribution must be discussed with the designers and the database administrator. Performance tests can be carried out either on a database including real data or on a mock database, but the database size should be compatible with the average expected data volume. On the other hand, stress tests are typically carried out on mock databases whose size is significantly larger than what expected. Standard database metrics –such as maximum query response time– can be used to quantify the test results. To advance these testing activities as much as possible and to make their results independent of front-end applications, we suggest to use SQL to code the workload.

Recovery tests enable testers to verify the DBMS behavior after critical errors such as power leaks during update, network fault, and hard disk failures.

Finally, security tests mainly concern the possible adoption of some cryptography technique to protect data and the correct definition of user profiles and database access grants.

4.5 Testing the Front-End

Functional testing of the analysis front-ends must necessarily involve a very large number of end-users, who generally are so familiar with application domains that they can detect even the slightest abnormality in data. Nevertheless, wrong results in OLAP analyses may be difficult to recognize. They can be caused not only by faulty ETL procedures, but even by incorrect data aggregations or selections in front-end tools. Some errors are not due to the data mart; instead, they result from the overly poor data quality of the source database. In order to allow this situation to be recognized, a common approach to front-end functional testing in real projects consists in comparing the results of OLAP analyses with those obtained by directly querying the source databases. Of course, though this approach can be effective on a sample basis, it cannot be extensively adopted due to the huge number of possible aggregations that characterize multidimensional queries.

A significant sample of queries to be tested can be selected in mainly two ways. In the "black-box way", the workload specification obtained in output by the workload refinement phase (typically, a use case diagram where actors stand for

Table 2: Coverage criteria for some testing activities; the expected coverage is expressed with reference to the coverage criterion

Testing activity	Coverage criterion	Measurement	Expected coverage
fact test	each information need expressed by users during requirement analysis must be tested	percentage of queries in the preliminary workload that are supported by the conceptual schema	partial, depending on the extent of the preliminary workload
conformity test	all data mart dimensions must be tested	bus matrix sparseness	total
usability test of the conceptual schema	all facts, dimensions, and measures must be tested	conceptual metrics	total
ETL unit test	all decision points must be tested	correct loading of the test data sets	total
ETL forced-error test	all error types specified by users must be tested	correct loading of the faulty data sets	total
front-end unit test	at least one group-by set for each attribute in the multidimensional lattice of each fact must be tested	correct analysis result of a real data set	total

user profiles and use cases represent the most frequent analysis queries) is used to determine the test cases, much like use case diagrams are profitably employed for testing-in-the-large in generic software systems. In the "white-box way", instead, the subset of data aggregations to be tested can be determined by applying proper coverage criteria to the multidimensional lattice[1] of each fact, much like decision, statement, and path coverage criteria are applied to the control graph of a generic software procedure during testing-in-the-small.

Also in front-end testing it may be useful to distinguish between unit and integration tests. While unit tests should be aimed at checking the correctness of the reports involving single facts, integration tests entail analysis sessions that either correlate multiple facts (the so-called drill-across queries) or take advantage of different application components. For example, this is the case when dashboard data are "drilled through" an OLAP application.

An integral part of front-end tests are usability tests, that check for OLAP reports to be suitably represented and commented to avoid any misunderstanding about the real meaning of data.

A performance test submits a group of concurrent queries to the front-end and checks for the time needed to process those queries. Performance tests imply a preliminary specification of the standard workload in terms of number of concurrent users, types of queries, and data volume. On the other hand, stress tests entails applying progressively heavier workloads than the standard loads to evaluate the system stability. Note that performance and stress tests must be focused on the front-end, that includes the client interface but also the reporting and OLAP server-side engines. This means that the access times due to the DBMS should be subtracted from the overall response times measured.

Finally, in the scope of security test, it is particularly important to check for user profiles to be properly set up. You should also check for single-sign-on policies to be set up properly after switching between different analysis applications.

4.6 Regression Tests

A very relevant problem for frequently updated systems, such as data marts, concerns checking for new components and new add-on features to be compatible with the operation of the whole system. In this case, the term *regression test* is used to define the testing activities carried out to make sure that any change applied to the system does not jeopardize the quality of preexisting, already tested features and does not corrupt the system performances.

Testing the whole system many times from scratch has huge costs. Three main directions can be followed to reduce these costs:

- An expensive part of each test is the validation of the test results. In regression testing, it is often possible to skip this phase by just checking that the test results are consistent with those obtained at the previous iteration.

- Test automation allows the efficiency of testing activities to be increased, so that reproducing previous tests becomes less expensive. Test automation will be briefly discussed in Section 6.

- Impact analysis can be used to significantly restrict the scope of testing. In general, *impact analysis* is aimed at determining what other application objects are affected by a change in a single application object [9]. Remarkably, some ETL tool vendors already provide some impact analysis functionalities. An approach to impact analysis for changes in the source data schemata is proposed in [14].

5. TEST COVERAGE

Testing can reduce the probability of a system fault but cannot set it to zero, so measuring the *coverage* of tests is necessary to assess the overall system reliability. Measuring test coverage requires first of all the definition of a suitable coverage criterion. Different coverage criteria, such as statement coverage, decision coverage, and path coverage, were devised in the scope of code testing. The choice of one or another criterion deeply affects the test length and cost, as well as the achievable coverage. So, coverage criteria are chosen by trading off test effectiveness and efficiency. Examples of coverage criteria that we propose for some of the testing activities described above are reported in Table 2.

[1]The multidimensional lattice of a fact is the lattice whose nodes and arcs correspond, respectively, to the group-by sets supported by that fact and to the roll-up relationships that relate those group-by sets.

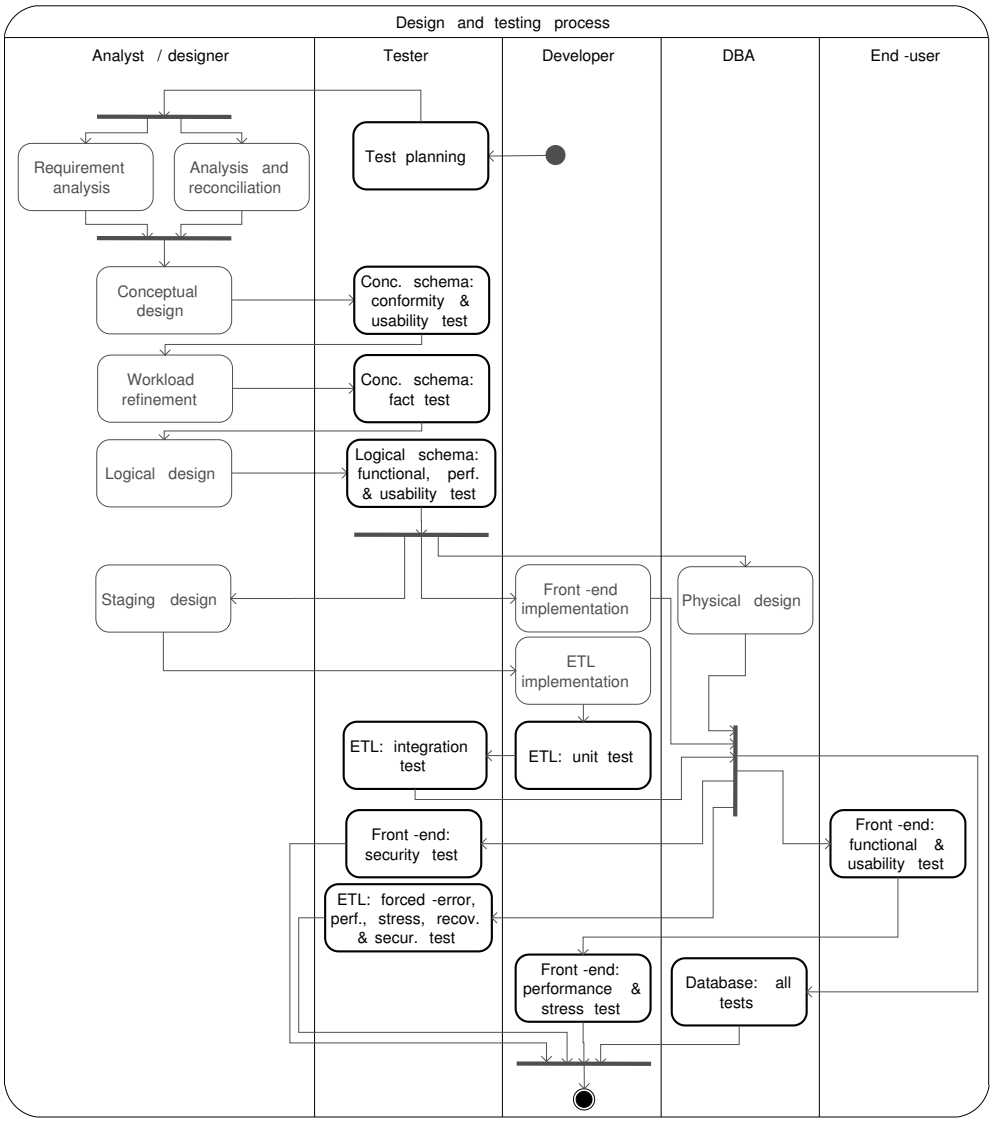

Figure 2: UML activity diagram for design and testing

6. A TIMELINE FOR TESTING

From a methodological point of view, the three main phases of testing are [13]:

- Create a test plan. The *test plan* describes the tests that must be performed and their expected coverage of the system requirements.
- Prepare test cases. *Test cases* enable the implementation of the test plan by detailing the testing steps together with their expect results. The reference databases for testing should be prepared during this phase, and a wide, comprehensive set of representative workloads should be defined.
- Execute tests. A *test execution log* tracks each test along and its results.

Figure 2 shows a UML activity diagram that frames the different testing activities within the design methodology outlined in Section 3. This diagram can be used in a project as a starting point for preparing the test plan.

It is worth mentioning here that test automation plays a basic role in reducing the costs of testing activities (especially regression tests) on the one hand, on increasing test coverage on the other [4]. Remarkably, commercial tools (such as QACenter by Computerware) can be used for implementation-related testing activities to simulate specific workloads and analysis sessions, or to measure a process outcome. As to design-related testing activities, the metrics proposed in the previous sections can be measured by writing ad hoc procedures that access the meta-data repository and the DBMS catalog.

7. CONCLUSIONS AND LESSONS LEARNT

In this paper we proposed a comprehensive approach which adapts and extends the testing methodologies proposed for general-purpose software to the peculiarities of data warehouse projects. Our proposal builds on a set of tips and sug-

gestions coming from our direct experience on real projects, as well as from some interviews we made to data warehouse practitioners. As a result, a set of relevant testing activities were identified, classified, and framed within a reference design methodology.

In order to experiment our approach on a case study, we are currently supporting a professional design team engaged in a large data warehouse project, which will help us better focus on relevant issues such as test coverage and test documentation. In particular, to better validate our approach and understand its impact, we will apply it to one out of two data marts developed in parallel, so as to assess the extra-effort due to comprehensive testing on the one hand, the saving in post-deployment error correction activities and the gain in terms of better data and design quality on the other.

To close the paper, we would like to summarize the main lessons we learnt so far:

- The chance to perform an effective test depends on the documentation completeness and accuracy in terms of collected requirements and project description. In other words, if you did not specify what you want from your system at the beginning, you cannot expect to get it right later.
- The test phase is part of the data warehouse life-cycle, and it acts in synergy with design. For this reason, the test phase should be planned and arranged at the beginning of the project, when you can specify the goals of testing, which types of tests must be performed, which data sets need to be tested, and which quality level is expected.
- Testing is not a one-man activity. The testing team should include testers, developers, designers, database administrators, and end-users, and it should be set up during the project planning phase.
- Testing of data warehouse systems is largely based on data. A successful testing must rely on real data, but it also must include mock data to reproduce the most common error situations that can be encountered in ETL. Accurately preparing the right data sets is one of the most critical activities to be carried out during test planning.
- No matter how deeply the system has been tested: it is almost sure that, sooner or later, an unexpected data fault, that cannot be properly handled by ETL, will occur. So keep in mind that, while testing must come to an end someday, data quality certification is an ever lasting process. The borderline between testing and certification clearly depends on how precisely requirement were stated and on the contract that regulates the project.

8. REFERENCES

[1] A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, and S. Paraboschi. Designing data marts for data warehouses. *ACM Transactions on Software Engineering Methodologies*, 10(4):452–483, 2001.

[2] K. Brahmkshatriya. Data warehouse testing. http://www.stickyminds.com, 2007.

[3] R. Bruckner, B. List, and J. Schiefer. Developing requirements for data warehouse systems with use cases. In *Proc. Americas Conf. on Information Systems*, pages 329–335, 2001.

[4] R. Cooper and S. Arbuckle. How to thoroughly test a data warehouse. In *Proc. STAREAST*, Orlando, 2002.

[5] P. Giorgini, S. Rizzi, and M. Garzetti. GRAnD: A goal-oriented approach to requirement analysis in data warehouses. *Decision Support Systems*, 5(1):4–21, 2008.

[6] M. Golfarelli, D. Maio, and S. Rizzi. The dimensional fact model: A conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 7(2-3):215–247, 1998.

[7] M. Golfarelli and S. Rizzi. *Data warehouse design: Modern principles and methodologies*. McGraw-Hill, 2009.

[8] D. Haertzen. Testing the data warehouse. http://www.infogoal.com, 2009.

[9] R. Kimball and J. Caserta. *The Data Warehouse ETL Toolkit*. John Wiley & Sons, 2004.

[10] J. Lechtenbörger and G. Vossen. Multidimensional normal forms for data warehouse design. *Information Systems*, 28(5):415–434, 2003.

[11] W. Lehner, J. Albrecht, and H. Wedekind. Normal forms for multidimensional databases. In *Proc. SSDBM*, pages 63–72, Capri, Italy, 1998.

[12] J. McCall, P. Richards, and G. Walters. Factors in software quality. Technical Report AD-A049-014, 015, 055, NTIS, 1977.

[13] A. Mookerjea and P. Malisetty. Best practices in data warehouse testing. In *Proc. Test*, New Delhi, 2008.

[14] G. Papastefanatos, P. Vassiliadis, A. Simitsis, and Y. Vassiliou. What-if analysis for data warehouse evolution. In *Proc. DaWaK*, pages 23–33, Regensburg, Germany, 2007.

[15] G. Papastefanatos, P. Vassiliadis, A. Simitsis, and Y. Vassiliou. Design metrics for data warehouse evolution. In *Proc. ER*, pages 440–454, 2008.

[16] R. Pressman. *Software Engineering: A practitioner's approach*. The McGraw-Hill Companies, 2005.

[17] M. Serrano, C. Calero, and M. Piattini. Experimental validation of multidimensional data models metrics. In *Proc. HICSS*, page 327, 2003.

[18] M. Serrano, J. Trujillo, C. Calero, and M. Piattini. Metrics for data warehouse conceptual models understandability. *Information & Software Technology*, 49(8):851–870, 2007.

[19] I. Sommerville. *Software Engineering*. Pearson Education, 2004.

[20] P. Tanuška, W. Verschelde, and M. Kopček. The proposal of data warehouse test scenario. In *Proc. ECUMICT*, Gent, Belgium, 2008.

[21] A. van Bergenhenegouwen. Data warehouse testing. http://www.ti.kviv.be, 2008.

[22] P. Vassiliadis, M. Bouzeghoub, and C. Quix. Towards quality-oriented data warehouse usage and evolution. In *Proc. CAiSE*, Heidelberg, Germany, 1999.

[23] Vv. Aa. Data warehouse testing and implementation. In *Intelligent Enterprise Encyclopedia*. BiPM Institute, 2009.