Test-Data Quality as a Success Factor for End-to-End Testing
An Approach to Formalisation and Evaluation

Yury Chernov
SIX Group Services AG, Selnaustrasse 30, Zurich, Switzerland

Keywords: Data Quality, Test-Data, End-to-End Testing, Data Quality Factors, Test-Data Quality Modelling.

Abstract: Test-data quality can be decisive in the success of end-to-end testing of complicated multi-component and distributed systems. The proper metrics allows to compare different data sets and to evaluate their quality. The typical data quality factors should be, on the one hand, enriched with the dimensions specific for the testing, and, on the other hand many requirements to the production system data quality become not relevant. The proposed formal model is based to great extent on the common sense and practical experience, and not over-formalised. The implementation requires quite an effort, but once established can be very useful.

1 INTRODUCTION

The aim of end-to-end testing is to verify the complete business functionality of a system starting from entry points and including the outgoing information to end-users. The functionality is defined by the requirements and corresponding use-cases. The end-to-end testing takes place on special test systems. Besides technical implementation of a test system one of the major points is the test-data. The quality of the test-data must ensure the testability and the business-relevant test coverage.

The definition and evaluation of the test-data quality is the topic of the current paper. In order to define the quality of the test-data we summarise the parameters of the data quality in general and then include specific points of the testing.

The idealistic aim would be to fully formalise the evaluation of the data-quality in order to enable a proper metrics and a quantitate comparison of one data-set to another. However, this aim is practically not reachable, or can be only partly reached. Therefore, our intention is to find a reasonable approach, suitable to the evaluation and helpful in finding the ways to improve the test-data. And the major question remains: how we can formally define the test-data quality.

The present paper is based on the testing experience by SIX Swiss Exchange (SSE) (http://www.six-swiss-exchange.com), where the requirements to the reliability and stability are very high. That is why the question of the test-data quality is extremely important.

2 DATA QUALITY DIMENSIONS

Although the presented discussion is rather generic: the rational and models are applicable to a wide range of data-intensive systems, the influence of the specific system under test, that is SSE, cannot be ignored. We will first discuss the system under the test especially in the context of the end-to-end testing and the standard data quality dimensions, and then introduce the test-data specifics reflected in the model.

2.1 System Under Test

SSE is highly complicated multi-component system. It includes a dozen of components running on different platforms, with different data bases (Oracle, SQL Server, MySql, etc.) and communication principles, protocols and technical means. The static or reference data that is defined in the dimension tables (Kimball, 2013) and dynamic or trading data (fact tables) arrives into the system through different interfaces starting from web- and application-based manual input and uploads, and including fast algorithmic-trading interfaces, where the time is differentiated in nanoseconds. The end-to-end test system should reproduce this landscape

Chernov, Y.
Test-Data Quality as a Success Factor for End-to-End Testing - An Approach to Formalisation and Evaluation.
DOI: 10.5220/0005971700950101
In *Proceedings of the 5th International Conference on Data Management Technologies and Applications (DATA 2016)*, pages 95-101
ISBN: 978-989-758-193-9

practically one to one.

The major specific of the system from the data point of view is that the data consumers (banks and other trading organisations) are to great extend data creators, since they issue the products that are traded on SSE. They are as well, as system customers, the major source of functional requirements. That is a very important point: we can refer to (Redman, 2013) who emphasised: *"I've noted repeatedly the importance of connecting data customers and data creators. One difficulty is that customers often speak in vague, confused, task-specific terms, while data creators need hard, tangible specifications on which to define and improve their processes."* However, we should not forget that this was stated for the real production data and we are investigating the data aspects in the end-to-end test system.

The data can be analysed from different points of view, which they use to call dimensions. The data quality approach is not homogeneous. It changes from dimension to dimension, since different aspects come in the foreground of the analysis. Therefore, it is important to discuss them, at least to point them out.

The information circulated in SSE includes the static data (issued products, participant information, technical connection data, billing parameters etc.) and dynamic data (trading information, calculated indices, statistical evaluations etc.). Both types of data come from the up-stream components and are integrated in the downstream components (data warehouse, monitoring and supervision tools, and information distribution utilities). The third data type is the information generated by the components themselves (information products), which is based on the incoming static and dynamic data. Of course that relates more to the downstream components, for instance, the spreads of the order books, or some daily statistic reports.

The data arrives into the system (incoming information), is stored and processed and distributed further to customers. The quality of these portions can be different. The problems and defects in incoming data can be partly corrected by the software that these data reads, verifies and stores.

The same process takes place for the outgoing data. When a certain data field is not populated for certain tuples, it may be set to an agreed value by the export routines.

But when the software itself has bugs (the testing actually is done to detect them), it can reduce the quality of the stored data by introducing the data problems.

From the point of view of the source the data can be external, that is comes from external sources outside of the systems (ex. issued financial instruments, indices form foreign stock exchanges, currency rates etc.), or internal. Internal data is produced either by the software that processes, consolidates and converts the data, or by internal users that have the role to enrich and maintain the data.

The character of the data can be business-relevant or pure technical one. The technical data can include, for instance, network configuration and user access paths. The quality of this data should be extremely high; otherwise, the system will simply not work properly.

Besides, the data relates to different objects in the system. The quality might be interesting for separate objects and even sub-objects. For instance, one of the objects is securities (products/instruments) listed on the stock exchange. As sub-objects we can consider different types, like shares, bonds, derivatives etc.

The summary of the data characteristics in SSE is presented in Tab. 1. Of course the data can be viewed from some additional aspects as well. However, we define those ones that are more relevant for the data quality modelling.

Table 1: Data Characteristics.

Data Dimension	Dimension Refinement
Type	Static
	Dynamic
	Generated
Information	Incoming
	Stored
	Outgoing
Source	External
	Internal
Character	Business
	Technical
Store	Temporal
	Long-term
Object	Products/Instruments
	Participants
	On-book trading data
	Off-book trading data
	……

The existing production SSE is reflected in end-to-end test environment. The major test environment includes 14 components: reference data repositories, trading engine, off-book trading component, monitoring system, data warehouse, data distribution system etc. It covers a big portion of real production functionality. It is never 100%, but it can be close to that.

2.2 Different Test Levels

In the current paper we focus our attention on the functional end-to-end testing. In the SSE landscape different test phases are implemented. Major of them are component and integration testing, which take place before the software is shipped to the end-to-end testing. The requirements on the test-data are different for different test phases.

The business relevance has lower priority for the component testing, but rather the combinatorics of possible data configurations is important. The aim of the component testing is to check all technically possible scenarios. That is done with synthetic data. This data is not coming from the up-streams components, but is stored in the specially prepared files. The testing is mostly automated. Especially the regression testing is appropriate, when the same input files feed the test components with different software versions and the results are compared.

For the integration testing the interface variants and protocol configurations are in focus. Here not only one separate component, but typically pares of components are serving as testees. However, the principle approach to the test-data is like in the component testing. The data can be just synthetic one. The closeness to the real production data is not very important. Important is to cover all possible protocol branches and all possible file configurations, when the communication between components is implemented through files. The last is typical for the reference data.

For the end-to-end testing the data must support all business functions and not necessary all technically possible variations. To fulfil that the data should be close to the real production one and also cover future needs, which are technically possible, but not activated yet. Therefore, the data is often a combination of production and synthetic portions.

2.3 Data Quality Factors

The traditional data quality factors or dimensions are defined in many publications (Redman, 2013; Batini, 2006; Rainardi, 2008). They are mainly oriented on the data warehouse concept, which is quite appropriate for us as well.

When we are speaking about the data quality we should not forget two aspects or objects, namely data model and data set. Usually they mean the last, when discussing the data quality. In the current work we are doing that as well, because from the testing point of view in general and the end-to-end testing in particular the data set is the major point of interest. The data model belongs primarily to the design. However, often only during the testing you detect the model problems and the feedback can be still very useful.

Below in Tab. 2 we list the principle quality factors. The list is rather generic and covers both data models and data sets. That is why these factors serve as a good basis for more precise and practical definition when implementing them to the test quality evaluation.

Table 2: Data Quality Factors.

Data Quality Factor	Explanation
Correctness	Accuracy: data are correct (data formats, defaults, initial load, NULLs etc.). Precision: data are adequately specific. Granularity: data are sufficiently detailed.
Completeness	Completeness: all data are available. No Gaps: all needed values are present. State: data are in usable state.
Consistency	No conflicting data.
Conformity	Conformity with critical rules.
Integrity	Identity: unique keys are defined. Referential Integrity. Cardinality: relationship constrains are present. Dependency: functional constrains are present. Uniqueness: no duplicate data.
Validity	Values are within required range. No Null values in Non-NULL fields.
Timeliness	Currency: data are up to date. Repetition: data include enough history. Continuity: no historical gaps. Sequentially: data are logically sequenced.
Data Comprehension	Data match business; data are understandable; data are easy to view; data are structured in logical sequence; data are fit-to-tool (easily imported/exported).

We are discussing these factors below. But first of all describe specifics of requirements to the test-data that define its quality.

2.3.1 Test-Data Quality

The data quality factors reflect the general data quality aspects. The test-data and especially the test-data for the end-to-end testing have its specifics:

- it should cover all real production cases and configurations
- it should properly reflect the real production data relations and stochastic distributions
- it should support possible future functionality that is not used currently in the production system, but is possible technically
- it should quantitatively reflect the requirements for non-functional testing, that is over the current production volumes.

These specifics do not influence the general approach, however, they definitely important for the evaluation in the model presented below.

The test-data quality has one additional characteristic. Namely, how good the data covers all needs of the planned test activity, that is how good it for the execution of all planned test cases. Finally that is the crucial thing about the test-data.

The test coverage is usually derived from business and technical use cases and requirements. Each use case is typically transferred into several test requirements. When you manage to build a pure hierarchal test planning system then each test requirement is covered by several test cases. In practise you may have more complicated relations – many-to-many, when one test case supports several test requirements. Besides, every test case may have several configurations.

The quality factor reflects how good this complicated structure is supported by the test-data, so, that every test case could be executed with all its configurations. This criterion complements the set of quality factors.

2.3.2 Correctness

The data accuracy is easily formalised both for the interfaces and in the data store. They check formats, default values, initial load values against system specifications, empty fields. We are speaking about pure technical checks. More business-oriented verification is done along the data comprehension dimension.

Precision for a data entity should be the same all over the system. It makes no sense to store a field with, say, six decimal places, when the data source delivers it with only four positions after decimal point. The check of the precision aspect is complicated, since the same variables might have different names in different components (see the consistency dimension) and the proper analysis requires a big analytical rather than formal effort.

The granularity is business-defined and is covered in the data model level.

For SSE the major focus lays on default values, initial load for new versions and migrations, and NULLs. The NULL-values are traditionally the source of many problems. They often play an important role in the integration between components when one component designed by an outsourcing company, say in India, allows NULLs for specific field, and the next component that has been purchased from a provider in Australia expects no NULLs on the interface. Often such problems happen. Generally NULL can note either non existing value, or existing but unknown (not provided by the data source), or when you do not know, whether the value exists.

2.3.3 Completeness

The data completeness is as well easily checked on the database level. It is better when you have the statistical reference from the real production. The source of the data in the test system is mainly test automation scripts. They should be configured in such a way that at least the data relations (not always volumes) remain production-like.

On the table level we should verify that there are no empty tables. For instance, in the current SSE data warehouse test system there are 36 empty tables out of totally 187. However, some of them relate to the functions, which are not tested in the end-to-end system, others are not populated any more, but still kept in production for historic reporting. There are some tables that have been introduced, however, not used in production, since the corresponding business functionality is not yet activated.

On the record level – check for empty, not-populated fields. Here we should verify not simply NULLs, but those fields that are filled in some components, but empty in other.

From the dynamic point of view it should be controlled that there is data for all days (in SSE context) or other relevant time entities. For instance, in the current test data warehouse there is data from 01.06.2015 till 01.05.2016, that is, 335 calendar days (2016 is a leap-year). That corresponds to 231 business days (minus weekends and bank holidays). In the data warehouse there are 220 loaded days, i.e. 11 days are missing for some reasons.

From the object point of view – all objects (like legal entities, participants or trading users by SSE) should include required data.

For the end-to-end test purposes it is necessary that the data is complete and has no gaps during the test cycles. Between the test cycles some data gaps are acceptable. However, we are trying to avoid that by scheduling the execution of automatic scripts every day. Besides, the data gaps can be added on purpose - to test, how the system copes with them.

2.3.4 Consistency

Formally conflicts may occur when the same data in different tables is imported from different sources. If their values are not the same, we can speak about conflicting data.

Another aspect of this problem is different names for the actually same data entities. Often that happens when the data entities are in different components. The optimal way is to build the data name maps. The formalisation of these maps enables the automation of the data consistency checks.

In SSE there are rather many disagreements in data field names due to historically development of components by different software suppliers.

2.3.5 Conformity

Data rules are implemented either in the database or on the application level - in GUIs or interfaces. The important point here is to check the historical data. A new implemented rule will not allow new non-conform data. However, the old one that had been entered into the system before can include non-conform values. Often the conformity errors are detected when a user tries to change these old data entities.

In SSE only for reference data components the conformity aspect of the quality is relevant, because trading components often receive just the current data (for the current business day). So even when they do have some conformity problem, the root cause is in the reference data components. The verification can be technically easily done on the databases. The difficulty is to consistently formalise all business rules, which are often implemented on the interface level, in terms of the database entities.

2.3.6 Integrity

That relates mainly to the data-model. Of course if the referential integrity, cardinality and dependency are not defined on the database level then they are the point for the formal verification on the data-set level. But the most interesting here could be the historical development of data-sets, when they grew incrementally from version to version with the data model changes. In this case the referential integrity and cardinality of the historic data could be broken.

The verification is analogous to the consistency dimension. The integrity rules are technical ones and formulated on the database level.

2.3.7 Validity

Validity in out context is about the data ranges. There are two types of required range. The first is the technical one, which is defined on the database level. These ranges should be checked, like in the case of integrity, only for historical data. The second type is business-relevant ranges. This means that technically the values outside the range are possible, but they are not meaningful from the business point of view. These ranges could be very interesting for the evaluation of the data quality of the end-to-end testing. For the component testing, especially non-functional one, the extreme values are needed.

The evaluation becomes complicated, since some rules have been changed. So the validation should be estimated based on the rules valid for the specific period.

2.3.8 Timeliness

The authors of (Batini, 2006) defined the currency formally as the age of the data unit plus difference between the delivery time (the time the information product is delivered to the customer) and the input time (the time the data unit is obtained).

Volatility is the length of time data remains valid.

Timeliness, which is measured from 0 (bad) to 1 (good), equals 1 minus relation of the currency to the volatility. However, this formal approach is not always relevant.

In the SSE end-to-end test system the new data is generated daily by means of automated scripts. The quality check includes the verification the daily process that is that all tables are populated in all components. By this we guaranty no historical gaps.

From time to time (two-three times a year) the test environment is set-up from scratch as a copy of production system. This copy includes just the reference data, so the system starts its live from zero. However, some applications like billing or statistical evaluations require much historical data. Therefore, the test environment must “live” for some time, before all tests can be executed.

2.3.9 Data Comprehension

The data comprehension dimension is least

formalised. Actually it can be evaluated only on the bases of end-user opinions. For the testing this dimension is probably not very interesting, unless the data comprehension requirements are more formally formulated.

3 DATA QUALITY MODEL

Above we presented just some explanations to the data quality factors. Let us abstract from the formal definition of them and assume that they can be evaluated somehow by experts or with special procedures. For instance, one of the aspects of completeness could be the number of not populated fields and the number of empty tables in a database. For us is important to compare different data sets to each other and not the formal quantitative definition of the quality level. This comparison might involve two variants of data sets for the same system under the testing or two states of the same data set at different points of time, the last can reflect, for instance, two different projects.

There are three known approaches to the definition of data quality dimensions: theoretical (Wand, 1996), empirical (Wang, 1996) and intuitive (Redman, 1996). We are trying to combine them in a simple model.

We define the quality model as a primitive weighted sum of the factor levels:

$$q = \sum_i^n a_i \cdot x_i \qquad (1)$$

Where

x_i is the level of factor i,
a_i is the weight of the factor i,
n – total number of factors

In our case n = 8 (Tab. 2), and x_1 reflects the level of correctness, x_2 – the level of completeness, etc. The model (1) is very generic, the exact definition the levels and the weights is application specific.

In order to provide compatibility, the test-data quality models must be mapped on the same scale. The most appropriate seems the normalised scale. Therefore, we can state that both x_i and q are defined on the scale of 0 to 1. The weights a_i should be defined accordingly to ensure the proper value range for q.

The definition of a_i is rather complicated. The only reasonable way seems to be an expert evaluation. However more formal procedures could be thinkable as well.

3.1 Structure-based Approach

As it was mentioned above, the SSE system includes many non-homogeneous components. The first approach to the data quality evaluation is to evaluate the quality levels for every component and then to aggregate them with a product function:

$$QS = \prod_j^m b_j \cdot q_j \qquad (2)$$

Where q_j is the quality level of j-th component, estimated by (1); b_j is the weight of the j-th component.

We should use the product function rather than a weighted sum, because the quality is hardly defined by a bottleneck component. Therefore, when the data quality of an up-stream component is poor, the high quality of a down-stream one cannot improve the total value.

3.2 Object-based Approach

Basic model (1) & (2) may be implemented not exclusively to the whole data structure, but only for selected objects as well. In our application we can take for instance just traded securities or only billing relevant data entities. This approach of course requires a proper analysis of the data and very precise business know-how. In other words it cannot be realised just formally, as it is in principle possible with the structure-based approach.

3.3 Use-case-based Approach

This approach we call “use-case-based”, but practically it is based on the test cases and their configurations. We assume that the above described chain use case –> test case –> test case configuration (test instance) covers all the needed requirements to the end-to-end testing. Of course that is not always the case and strongly depends on the quality of the test specifications and plans. However, in SSE that should be assumable. In general, if the coverage of the system functionality by the testware is not good, then the test-data quality cannot add a lot.

We can detect, whether the test-data set supports a test instance, either by trying to execute it or just ask an expert (tester). By the end we have binary value: true or false. Let us call this variable “test instance quality indicator” and denote it through t_{ij} for the j-th instance of the i-th test case. The test-data quality may be modelled by the following expression:

$$QU = \frac{\sum_{i,j}^{T,k} c_{ij} \cdot t_{ij}}{\sum_{i,j}^{T,k} c_{ij}} \quad (3)$$

Where T is the number of test cases, k - number of configurations, and c_{ij} is the corresponding weight of the test case. Number of configurations is normally individual for every test case; therefore, we should actually speak about k_i.

The evaluation with the model (3) is usually effort and time consuming. However, once done, it could be very useful. It presents a different view on the test-data quality. For instance, as we already mentioned above, the data warehouse in the end-to-end test system of SSE has 36 empty tables out of totally 187, although they are populated in the production system. That has rather strong influence on the data quality evaluation according to (1)-(2). However, all of end-to-end test cases can be executed, and the value of QU from (3) could be high.

4 CONCLUSIONS

The dimensions that are typical for the data quality analysis can be applied to the test-data for the end-to-end testing as well. They provide useful information. However, from the one hand, the typical dimensions should be enriched with the specific for the test-data factors. From the other hand, not all requirements, which are important for the real productive data, are relevant for the test-data. Therefore, the corresponding model should be based not only on the formal approach, but on the common sense and empirics as well.

The practical results require a big effort. However, once established and implemented in terms of models (1) - (3), the procedure can be very effective in evaluation of the test-data quality.

Very challenging task is the design of a more formal model for the proper definition of the weights in (1), (2), and (3). Till know they are defined intuitive or partly by experts and are not very differentiated. The proper approach seems to be the combination of expert estimation with several simulations of the test data.

REFERENCES

Batini, C., Scannapieco, M., 2006. Data Quality: Concepts, Methodologies and Techniques, Springer. Berlin.

Kimball, R., Ross, M. 2013. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. Wiley, 3rd Ed.

Rainardi, V., 2008. Building a Data Warehouse: With Examples in SQL Server, Apress. New York.

Redman, T.C., 1996. Data Quality for the Information Age. Artech House.

Redman, T.C., 2013. Data Quality Management Past, Present, and Future: Towards a management System for Data. In. Handbook of Data Quality: Research and Practice (Ed. Sadig, S.), Springer.

Wand, Y., Wang, R.Y. 1996. Anchoring Data Quality Dimensions in Onto-logical Foundation. Communication of the ACM, 39, 11.

Wang, R.Y., Strong, D.M. 1996. Beyond Accuracy: What Data Quality Means to Data Consumers. Journal of Management Information Systems 12, 4.