Testing Big Data

(Assuring the Quality of Large Databases)

Harry M. Sneed
Software Quality Assurance Team
ZTP-Prentner-IT
Vienna, Austria
Harry.Sneed@T-Online.hu

Katalin Erdoes
Tool Development Team
SoRing Kft.
Budapest, Hungary
Erdoes.Katalin@T-Online.hu

***Abstract*— The volume and variety of modern day databases presents a particular challenge to the system testing community. The question is how to go about testing such large collections of various data types ranging from tables to texts and images. To test those applications which use them, these conglomerations of multiple data object types have to be automatically generated and validated. There is no other way but to automate the test process. This contribution outlines the challenge and presents an automated approach to setting up and testing big data bases. At the end a case study of a large data warehouse is discussed with lessons learned from that industrial test project.**

Keywords—mixed databases, relational data, text data, images, data warehouses, validation rules, data assertions, data testing tools;

I. SEPARATION OF FUNCTION AND DATA

Due to the limitations of core storage in the early days of computing, large sets of data had to be moved out to external storage media, first to magnetic tapes and later to disc drives and other mass storage devices. The price for this separation of data from functionality was the long access time required to move data back and forth from main storage to the periphery storage. The data access time determined the speed in which programs could be executed. Reducing this time has been a major topic of computer technology research going back to the 1960s. Only recently with the advent of In-Memory data storage through such systems as Hadoop from the SAP does a solution appear to be in sight. However, as of now these systems also have their limitations so it still remains necessary to maintain large volumes of data offline on external storage devices [1]. Up until now software systems have been tested by manually filling out the data files and tables they use with character and numeric data copied over from other sources or typed in by users of that data.

2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)
13th User Symposium on Software Quality, Test and Innovation (ASQT 2015)
978-1-4799-1885-0/15/$31.00 © 2015 IEEE

The persons responsible for the data had to know which data values in what combinations were required to demonstrate that the target system functions properly. This of course required domain knowledge on the part of the data testers. Another approach was to generate random data based on the data type descriptions, i.e. the database schemas, but random data leads to random results which cannot be verified. Random data is only good for load and performance testing. It is not good for testing a system's functionality [2]. For that specific data values are required which match to the algorithms contained in the system and to the data submitted by the end user. This problem has never been adequately solved and now come the big databases with huge volumes of multiple data objects – long character strings, various numeric types, large texts, codes images, pictures and even films. Conventional testing approaches are no longer able to cope with the data volume, let alone for the data variety. Other ways of setting up the data before testing and confirming the correctness of the data after testing have to be found [3].

II. FORMATED VS. NON-FORMATTED DATA

From the very beginning database developers have been confronted with the problem of distinguishing between formatted data and text data. Formatted data contains character strings and numbers with a fixed length. Each data value is a field in a record or a column in a table. It can be a short text or a number. Later longer texts with a variable length were added but the values remained discrete and could be accessed individually in a relational database. In non-relational databases the data values are the attributes of a data object. They are identified by their position within the object or record. Not the individual values but the object as a whole is stored and retrieved. However, the objects are formatted and the individual values can be extracted for processing [4].

This is not the case with unformatted data. Unformatted data consists of variable length texts with no internal structure. They have to be parsed to recognize what is in them. Today one speaks of NOSQL databases, but such databases have been around for a long time. Already in the 1970's formatted and non-formatted databases existed next to one another. For

instance Siemens offered the database system SESAM for storing and retrieving formatted data in tables, and the database system GOLEM for storing and retrieving unformatted data in text documents. The two database types were kept strictly separated. They were even processed by separate application types [5]. This is no longer true today. Today the two database types are merged. The same application can access formatted and non-formatted data at the same time. For testing purposes, that means data generators have to be able to generate both representative discrete values and representative texts within the same test database. On the other hand, data validators have to be able to not only confirm the correctness of discrete data values – numbers, strings and codes - but also the correctness of whole texts consisting of lines and words. This is a new challenge to test automation. (see Figure 1: Dual Database Systems)

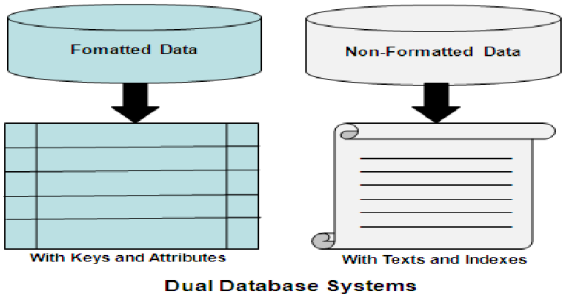

Figure 1: Separate relational and text databases

III. STORING RULES AS DATA

In addition to the unformatted and formatted data modern databases also contain rules for checking and processing the data, so called stored procedures. It is disputable whether or not such algorithms should be stored together with the data, but if they are there they have to be tested. This presents yet another challenge to the testers. The database tester must ensure that every rule is triggered and that it does what it is supposed to do. For example, if there is a rule to check the content of a particular data column and produce a warning if an illegal value appears there, then values must be assigned to that column which will trigger the warning. In testing manually, the tester will have to know the rules and assign data values to trigger them. In an automated test, the test data generator must be able to parse and interpret the rules in order to generate the required data values to test them [6].

Such data rules are at the heart of what is today referred to as business intelligence [7]. They are contained in big data bases to ensure the integrity of the data stored therein, which means testing big databases also means testing the rules of those databases. Since testing them manually can be very time consuming, it is recommended to automate their test, especially if there are many such rules. This presents a major challenge to test tool developers since the rules have to be processed alongside the data. The case study at the end of this paper deals with the automation of database rule checking.

IV. EMERGENCE OF DATA WAREHOUSES

In the course of time, the relational databases became increasingly large and powerful with many related tables. In the 1990's a new term came into existence to denote a network of related databases with built-in reporting capabilities – the data warehouse [8]. All the data of an entire enterprise are collected together in a common network of linked data tables. This makes it possible for applications to navigate through the data going from one table to another picking up bits of data along the way. This presents yet another challenge to the testers. They must synchronize the data in the various related tables. At the top of a data warehouse are master tables consisting of key terms pointing to associated values, i.e. something like tables of content, pointing to where data is stored. In addition to these complex data structures, there are ready-made programs stored in the database which produce standard data reports on a regular basis triggered by date and time, or upon demand by the user. Thus, the user is kept informed on the status of his data.

For the user this means that the database is taking over much of the functionality which the user previously had to program himself. For test automation it means that the test data generator must be able to generate data values in one table based on the values in other tables. Logical data test cases transcend table and database boundaries. In similar fashion, the data validator must check the values in one table depending on the data values of another table. There may be dependencies between any numbers of different data tables. So, one test case will consist of references to many data columns in many data tables. It becomes more and more difficult to specify such complex test cases.

The case study included in this paper describes testing such a large data-warehouse system. The author was required to create test cases for validating the contents of the data warehouse based on a specification of more than 5000 data rules. Some 300 related tables with more than 11,000 data attributes had to be validated. This case is typical for test projects concerning big data warehouses. (see Figure 2)

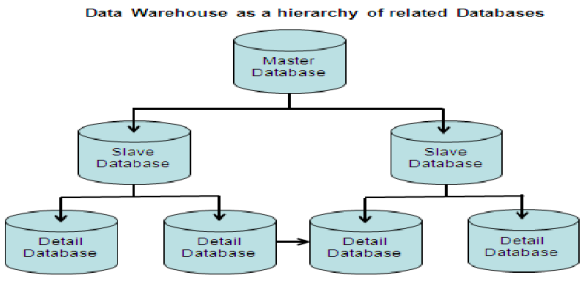

Figure 2: Data Warehouse = Network of related Databases

In the meantime data warehouses have evolved into so called "Business Intelligence" systems. With such systems it is possible to scan through thousands of data tables and data

objects to recognize data states which may be relevant to the user. As a prerequisite the user has to specify rules on what is relevant, for instance what combination of data values might indicate a potential threat to the business. Of course, these rules have to be tested to show that they can be relied on. A meta rule specification language is required to test the validity of the rules such as the one described in the case study.

V. INTEGRATION OF DATA, TEXT AND IMAGES

In the last years databases have been extended to include next to data and text also images such as diagrams and RFID codes. They are included as blobs within relational tables or kept in separate files which are referred to by pointers stored in the data tables. The databases are organized as networks of interrelated objects [9]. Some objects are conventional formatted data tables, others are unformatted texts and still others are graphical images. The challenge of testing is to create test databases containing representative objects which are related to one another. It is not enough just to store an image. That image must fit to the data which refers to it and to the text which explains it. Thus, within one test case, different data object types can be accessed and joined together. Joining takes on a new meaning. When generating such databases the tester has to refer to different sources of data. Thus, even before beginning to build up a database, he must first collect objects - tables, texts and images – which fit together. This is something that cannot be readily automated. It requires human judgment. (see Figure 3: Mixed Databases)

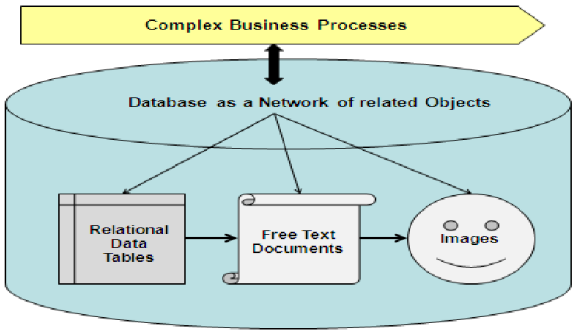

Figure 3: Databases as networks of stored Objects

Even more difficult than creating such heterogeneous databases is their validation. The goal here is to ensure that the various data objects really fit together. The text must fit to the images and both must fit to the data referring to them. For this complex semantic rules have to be defined which include key terms contained within the texts, characteristics of images and values of data attributes. Formulating such rules becomes a major task of the tester. It is not enough just to navigate through the database and confirm that all of the links are satisfied. The test must demonstrate that the links refer to the correct objects and that these objects are consistent with their usage. The texts should include the specified key terms and the images should have the correct characteristics. The tester must take on the role of an intelligent user [10].

VI. CURRENT STATUS OF BIG DATABASES

Current databases have evolved from storing and retrieving data to storing and retrieving things. They have become containers of things. They are similar to supermarkets which offer everything from food and drinks to clothes and household articles. Setting up such a database is a major challenge. Not only do the databases contain a wide variety of data objects, but the size of the databases has reached new dimensions. They have gone from gigabytes to giga-objects. There is practically no limit to their size.
The three main characteristics of big data are

- Volume
- Variety und
- Velocity [11].

With volume is meant the sheer size of the databases. Due to the possibility of distributing the database across multiple data servers there are no longer any hardware boundaries. The sky is the limit. One can store as many object and data instances as one can imagine.

With variety is meant the different types of data which can be stored within a single data container, everything from discrete numeric and string values to texts and images and on to video films and audio recordings. All of this can be stored and retrieved in various sequences and combinations.
With velocity is meant the speed with which the objects can be retrieved and put together. The search algorithms are constructed in such a way that many multiple search paths are executed parallel to one another. In the end the results of the different searches are joined together to form a consistent whole. Testing such a retrieval process requires that the testers understand how the search algorithms work and are able to predefine the results as a complex object If the tests are to be repeatable the post conditions must be specified in an interpretable language. Testers become abstract programmers whose job in testing the retrieval algorithms is just as difficult as writing the algorithms themselves. Measuring the performance of big databases is the least problematic [12].

VII. PROBLEMS WITH DATA QUALITY

The existing databases of most IT users are the product of a long evolution. It began with their conception by a database designer on the basis of the information available to him at that time. This was followed by the hand coding or the automatic generation of a database schema reflecting the design. After that the database was installed and applications began to use it. As new applications came along the structure was altered or extended to accommodate them. These structural changes were often made in an ad hoc manner, so that the complexity of the structure grew and the quality sank, much like software systems according to the laws of software evolution. In addition to this structural deterioration, the contents of the database became corrupted by erroneous

programs storing incorrect values and deleting essential records. With large databases it is difficult to recognize such quality erosion but over time it spreads like a cancerous infection causing more and more system failures. Thus not only program quality but also data quality suffers under erosion [13].

VIII. CASE STUDY IN TESTING A BIG DATABASE

A. The Data Specification

In the case described here the database involved was a data warehouse based on an Oracle relational database with 266 data tables and close to 11,000 data attributes. The data warehouse was intended to contain all the data delivered by the branch offices to an international bank. The data was delivered in CSV files with attributes corresponding to those in the data warehouse. Those files had to first be filtered, checked and converted before being loaded into the warehouse. For converting the incoming data some 5,300 rules were specified in an Excel table indicating the data name and type and the transformation rules. The rules were formulated in a semi-formal English specification language with seven rule types.

- Rule Type 1: assignment of a variable value from the input file.
- Rule Type 2: assignment of a constant value.
- Rule Type 3: assignment of a value from a list of alternate values (enumeration).
- Rule Type 4: assignment of a compound value concatenated from several input variables.
- Rule Type 5: assignment of a compound value joined from several input files (Join).
- Rule Type 6: assignment of the result of an arithmetic function performed on several input variables and constants .
- Rule Type 7: assignment of the result of a standard procedure .

Each of these assignment rules could be conditional or unconditional. Conditional rules were dependent on some English language conditional statement such as ” if, when, as long as, in case, etc”. The conditional operands were either input variables or constants.

Unfortunately, the assignments and their conditions were formulated in a slightly different mode throughout the rule table. However, they were similar enough to be parsed and recognized. The great majority - some 3950 - could be processed as they were. The remainder had to reformulated in a semi formal syntax. The syntax was as follows

- assign <Filename>. <Attribute> for a 1:1 assignment of an input value
- assign ”<Constant>” for a 1:1 assignment of a constant value
- assign ”<constant>”!”<constant>” for a set of alternate values
- assign Table.Attribute | ”<constant>” | Table. Attributen for a concatenation assignment
- assign join Table_A.Attribute_A1 | Table_A.Attribute_A2 with Table_B.Attribute_B1 | Table_B.Attribute_B2 for concatenating values from different data sources
- assign Table_A.Attribute_A1 + Table_B.Attribute_B1 * 2; for arithmetic expressions. There was no nesting of clauses nor precedence rules here, so the arithmetic expression was resolved in a simple left to right sequence
- assign Function (Param_1, Param_2, Param_3) whereby the parameters could be attributes in source input files, i.e. Table_C.Attribute_C1, or constant values, i.e. ”10.5”

With these assignment expressions, enhanced by an if expression in the form

if (Table_A.Attribute_A1 <oper> <operand>)
whereby <oper> := , < , >, <=, >=, <>, etc. and
<operand> = <Table>.<Attribute> or ”<constant>”

more than 4700 rules could be resolved automatically and converted into post condition assertions, which could then be tested. Of these rules some 750 were manually adjusted requiring more than 150 hours of effort. The fact that not more mapping rules were adjusted was due not only to the limited time available for the test, but also to the informal nature of the rules. Some rules were formulated in such a confusing manner which defied reformulation. There was simply no way to formalize them. It was a fault of the project that the rules were not properly defined to begin with. Had they been formulated at least in some semiformal form, it would have been possible to automatically process them immediately without having to spend valuable tester time in rewriting them. *(see Sample 1)*

Sample 1: Extract from a modified Rule table

```
INTEREST_RATE: "Link to TAB_X.ATTR_D interest conditions.
Debit interest conditions applicable to the account.";
" ? assign TAB_Y.ATTR_D | TAB_Y.ATTR_C |
TAB_X.ATTR_C | 'D' if TAB_Y.ATTR_D (debit) <> '0',
   assign TAB_X.ATTR_C if TAB_X.ATTR_D <> '0',
   assign TAB_X./ATTR_C | TAB_X.ATTR_D if
TAB_X.ATTR_B <> '0',
   assign '*na*' if TAB_Y.ATTR_D = '00' and TAB_X.ATTR_S =
'0' and TAB_X.ATTR_B = '00'
   assign TAB_X.ATTR_N|ATTR_Y|ATTR_O|ATTR_A if
other.(comment).";
```

B. Tools used to test the Data Warehouse

For the datawarehouse test four tools were required.

- **GenTest** for transforming the mapping rules into assertion procedures
- **GenSQL** for generating SQL procedures from the assertion procedure
- **AsrtComp** for checking and compiling the assertion procedures
- **DataVal** for validating the database contents

GenTest had the task of reading the rule table, extracting the rules, parsing them and converting them into assertions which could then be compared. It generated an assertion script for each target database table. As a byproduct GenTest generated test case specifications for the HP Test-Directory. These were stored in a central test repository. Since this was done automatically, there was no extra cost to the project. Finally it produced a statistical report on the number of entities, attributes and rules processed. This was important for measuring the degree of data coverage. *(see sample 2)*

Sample 2: Generated Test Script

```
file: ACCOUNT;
// This comparison procedure assumes that the old file contains the
following attributes:
//      from TAB_X Table: A_ID, A_TYPE, ATTR_S, ATTR_R, ATTR_C,
ATTR_S, ATTR_D, ATTR_F, ATTR_N
//      from TAB_Y Table: ATTR_C, ATTR_D, ATTR_E, ATTR_F,
ATTR_P
//      from TAB_Z Table: ATTR_R
if ( new.A_ID = old.ATTR_ID );
   assert new.A_ID = old.A_TYPE if (old.A_TYPE = "G");
   assert new.A_ID = "S" if (old.ATTR_A = "R" & old.ATTR_S = "S");
   assert new.A_ID = "C" if (old.ATTR_A = "R" & old.ATTR_S = "C");
   assert new.A_TYPE = "L" if (old.ATTR_A = "R" & old.ATTR_S = "L");
   assert new.A_RATE = old.ATTR_C if (old.ATTR_B = "0");
   assert new.A_INTEREST = old.ATTR_D;
   assert new.A_LIQU = old.ATTR_E;
   assert new.A_ACCT = old.ATTR_P;
   assert new.A_PC   = old.ATTR_PC;
   assert new.START_DATE = "2005.05.15" if (old.ATTR_H <> "0");
   assert new.REVIEW_DATE = old.ATTR_V if (old.ATTR_F <> "0" &
old.ATTR_N <> "0") ;
   assert new.REVIEW_DATE = "NULL" if (old.ATTR_F = "0") ;
   assert new.PRIMARY_INDICATOR = "P" ! "S" ! "*n.a.*";
   assert new.INDICATOR = "inactiv" if (old.ATTR_R = X"3") ;
   assert new.INDICATOR = "inactiv" if (old.ATTR_R = X"1") ;
   assert new.INDICATOR = "closed" if (old.ATTR_R = "C") ;
   assert new.INDICATOR = "active" if (other);
end;
```

GenSQL was a follow-up tool to GenTest. It parsed each generated assertion procedure to determine which attributes were required from what tables. It then generated two sets of SQL query procedures, one for selecting data from n input tables and joining them together to create a single CSV file for each data warehouse table, and another for selecting data from the target data warehouse table and down loading it into a CSV file. These two CSV files were to be the inputs to the data validation.

AsrtComp is a tool for checking the syntax of the assertion procedures, which could also be manually edited, and compiling them into intermediate symbol tables. There can be five tables for each entity to be validated:

- a header table for identifying the objects to be validated and the number of entries in each table
- a key condition table with the names and types of keys to be matched
- a comparison table assigning which new attributes are to be compared with which old attributes and/or constants
- a constant table containing an entry for each constant value used as an operand in the assertions
- a condition table containing all of the conditions to be fulfilled for the assertions to be executed. A pointer links the conditions to the assertions to which they apply [14].

DataVal is the final tool in the set. After reading the symbol tables it first processes the old CSV file, i.e. the inputs, and stores the values into a temporary database with their keys as an index. It takes the attribute tags from the first line of the CSV file and subsequently counts the columns to associate the values with the tags. It then processes the new CSV file, i.e. the outputs, and matches each new record by key to an existing old record. If a match is found the contents of the new record are compared with the values of the old record or with the constant in the symbol table or with computed values or with concatenated values or with a set of alternate constant values or with the lower and upper bounds of a range, depending on the conditions. Thus, there are many ways to verify the correctness of an output value. If no match is found, the old record is considered to be missing. After processing all new records, a second search is made of all the old records in the temporary database to see if they were compared or not. If not they are considered to be missing in the new file. A protocol lists out all of the incorrect data values, i.e. those that do not comply with their assertions, as well as all missing records. A set of test statistics summarizes the percentage of records missing and incorrect attributes. (see Sample 3)

Sample 3: Data Validation Report

```
+-------------------------------------------------
| File/Table Comparison Report
| Key Fields of Record(new,old)
+-------------------------------------------------+
| New: ACCOUNT
| Old: Attributes
+-------------------------------------------------+
| RecKey:100000USD114601001
| duplicate key in old File/Table
+-------------------------------------------------+
| RecKey:100000XXX104501001
| New: ATTR_ID             = G
| Old: Constant_Value      = L
+-------------------------------------------------+
| RecKey:100000YYY104501001
| New: ATTR_C              = XXX00
| Old: ATTR_C              = 0
+-------------------------------------------------+
| RecKey:100000ZZZ104501001
| New: ATTR_D              = 0
| Old: ATTR_P              = 1
+-------------------------------------------------+
| RecKey:100000ZZZ104501001
| New: REVIEW_DATE         = 9999-08-01_00:00:00
| Old: Constant_Value      = NULL
+-------------------------------------------------+
| RecKey:100000ATS196501100 is missing from the new
+-------------------------------------------------+
|Total Number of old Records checked:        91
|Number of old Records found in new File:    08
|Number of old Records with duplicate Keys:  72
|Number of old Records not in new Table:     11
|Total Number of new Records checked:        59
|Number of new Records found in old File:    08
|Number of new Records with alternate Keys:  00
```

```
|Number of new Records not in old File:       51
|Total Number of Fields checked:              93
|Total Number of non-Matching Fields:         46
|Percentage of matching Fields:               51 %
|Percentage of matching Records:              14 %
+-------------------------------------------------+
```

C. Data Warehouse Test Process

The process for testing the data warehouse system was well defined and automated. It consisted of 8 steps of which 4 were fully automated and 4 semi-automated (see Figure 4).

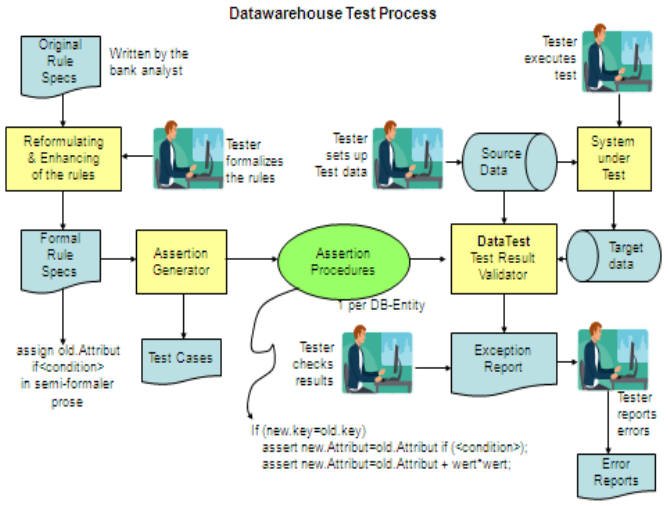

Figure 4: Data Warehouse Test Process

- Step 1: The mapping rules which did not comply with the language standard had to be reformulated. This was the most time consuming task of the whole process and could have been avoided if the rules have been properly specified in the first place.
- Step 2: The mapping rules were automatically converted into assertions by the GenTest tool
- Step 3: The SQL procedures were automatically generated from the assertions by the GenSQL tool
- Step 4: The input attributes for each target datawarehouse table were selected from the input tables, joined and downloaded into a CSV file
- Step 5: The output attributes for each target datawarehouse table were downloaded into a CSV file
- Step 6: The assertion procedures were compiled by the tool AsrtComp
- Step 7: The input and output CSV files were matched and the contents of the output file verified against the post assertions by the tool DataVal.
- Step 8: The testers examined the data validation results and reported any errors

In the end a report came out with the data errors which were then fed into the error tracking system by the testers. Once the rules had been reformulated the whole testing process could be repeated within a day. Normally such a test cycle would require at least 10 days. So the automation lead in this case to a significant improvement in test productivity [15].

IX. CONCLUSIONS AND FURTHER RESEARCH

Testing big data requires new processes and a higher degree of automation. The sheer volume of data together with the variety of the data makes it impossible to test manually. Testers need automated tools to scan through the mass of data and perform checks on the validity and consistency of the content. However to do this, rules must be defined which only a human being with domain knowledge can make. This means that testing big data becomes a job for data content experts, who can specify relations between data items and data objects and who can judge whether the images fit to the text and the text to the descriptors. Testing shifts from being a quantitative to a qualitative challenge. There is still much research required to answer that challenge [16].

REFERENCES

[1] D. Deroos, T. Deutsch, . C.Eaton, G. Lapis, P. Zikopoulos; Understanding Big Data – Analytics for Enterprse Class Hadoop and Streaming Data, McGraw-Hill Companies, New York, 2012

[2] C. Thomsen: „Technologie-Exkurs „Big Data“ – Hadoop, NoSQL, Text Analytics an Stream Computing“ Objektspektrum, Nr. 9, Mai, 2014, p. 8

[3] K. Wähner: „Realisierung intelligenter Geschäftsprozesse dank Big Data“, Objektspektrum, Nr. 9, Mai, 2014, p. 13

[4] E. Wolff, H.-C. Gürsoy:, A. Hartmann, K. Spichale: „NoSQL Architekturen – Wie sich die neuen Datenbanken auswirken“, Objektspektrum, Nr. 9, Mai, 2014, p. 34

[5] F. Gebhardt, „Semantisches Wissen in Datenbanken – Ein Literaturbericht“, Informatikspektrum, Nr. 10, 1987, p. 79

[6] J. Martin: DB2 – Concepts, Design an Programming, Prentice-Hall, Englewood Cliffs, 1989

[7] Scheer, A.-W., Hars, A.: “Extending Data Modelling to cover the whole Enterprise”, Comm of ACM, Vol. 35, No. 9, p. 166

[8] Kroenke, D.: “Beyond the Reltional Database Model”, IEEE Computer,May, 205, p. 89

[9] Liggesmeyer, P., Dörr, J., Heidrich, J.: “Big Data in Smart Ecosystems”, Informatikspektrum, Vol. 37, No. 2, 2014, p. 105

[10] Kaner, C. / Bach, J. / Pettichord, B.: Lessons learned in Software Testing, John Wiley & Sons, New York, 2002, p. 111

[11] Schermann, M., Krcmar, H.: “Big Data – Eine interdisziplinärische Chance für die Wirtschaftsinformatik”, Wirtschaftsinformatik, No. 5, 2014, p. 281

[12] Fischer, S.: “Big Data – Herausforderungen und Potenziale für deutsche Softwareunternehmen”, Informatkspektrum, Vol. 37, Nr. 2, 2014, p. 112

[13] Tayi, G., Ballou, D.: “Examining Data Quality”, Comm of ACM, Vol. 41, No. 2, p. 54

[14] Kaplan, D./Krishnan, R./Padman, R./Peters, J.: "Assessing Data Quality in Accounting Information Systems" in Comm. of ACM, Vol. 41, No. 2, Feb. 1998

[15] Sneed, H.: “Testing a Data Warehouse – An industrial Challenge”, IEEE Proc. Of TAICPART Workshop, Windsor, G.B., August, 2006, p. 203

[16] Yong, J.K./Kishore, R./Sanders, G.L.: “From DQ to EQ – Understanding Data Quality in the Context of E-Business Systems”, Comm. Of ACM, Vol. 48, No. 10, Oct. 2005, p. 75