AN EVALUATION OF OPEN SOURCE UNIT TESTING TOOLS SUITABLE FOR DATA WAREHOUSE TESTING

Robert Krawatzeck, Chair of Business Information Systems I, Chemnitz University of Technology, Germany, robert.krawatzeck@wirtschaft.tu-chemnitz.de

Anja Tetzner, Chair of Business Information Systems II, Chemnitz University of Technology, Germany, anja.tetzner@wirtschaft.tu-chemnitz.de

Barbara Dinter, Chair of Business Information Systems I, Chemnitz University of Technology, Germany, barbara.dinter@wirtschaft.tu-chemnitz.de

Abstract

Verification and validation are two important processes in the software system lifecycle. Despite the importance of these processes, a recent survey has shown that testing of data warehouse systems is currently neglected. The survey participants named besides others modest budget and the lack of appropriate tools as potential reasons for this circumstance. In order to verify these reasons, the paper at hand presents an evaluation of unit testing tools suitable for data warehouse testing. To address the modest budget problem, the range of evaluation candidates is limited to no charge, open source solutions, namely AnyDbTest, BI.Quality, DbFit, DbUnit, NDbUnit, SQLUnit, TSQLUnit, and utPLSQL. The evaluation follows the IEEE 14102-2010 guidelines for evaluation and selection of computer-aided software engineering tools in order to guarantee benefits from a practioners' as well as scientific point of view. It results in a detailed overview of how the testing tools meet criteria such as different testing functionalities. At least one tool, namely DbFit, can be identified as a promising candidate with regard to the requirements.

Keywords: Data Warehouse, Testing, Unit Testing, Automated Testing, Regression Testing, Tools, Frameworks, Evaluation, Open Source, IEEE 14102-2010.

1 INTRODUCTION

A data warehouse (DWH) is according to the well-known definition by Inmon (1996) a subject-oriented, integrated, time-variant, nonvolatile collection of data in support of management's decision-making process. Hence, a DWH is an information system which refers to special kind of database representing a "single point of truth" for organizational decisions. This purpose implies especially challenges with regard to the correctness of the presented information, since inaccurate information may lead to far-reaching consequences for an organization. On the one hand they might result in substantial financial losses (Manjunath et al. 2011), on the other side the trust of users in the DWH or in other information systems using the DWH as an information source might decrease due to such misinformation (Haertzen 2012). In addition, the correctness of DWHs is challenged by a rapidly changing environment and increasing competitive pressure which forces organizations to adapt their DWH in an agile and flexible manner (Wang et al. 2009). These adaptions may lead to defects such as an inconsistent database schema. In order to detect such defects, the correctness of DWH systems should be verified and validated after adaptations.

The verification and validation of the correctness of software systems are explicit processes within the software system lifecycle (ISO, IEC, IEEE 2008). The software *verification* process tests the correct implementation of the specified systems requirements of the whole system or system parts (ISO, IEC, IEEE 2008), and the software *validation* process tests the correct implementation of the specific intended use of the whole system or parts of it (ISO, IEC, IEEE 2008). The combination of both processes is often also referred to as 'software testing'. In addition to the distinction of what is tested (requirements vs. intended use), software testing can be further differentiated by the level of detail of the test target (Abran et al. 2004): *Unit testing* or *component testing* focuses on the system independent verification and validation of single software components, *integration testing* focuses on the correct functional interaction of those components, and *system testing* aims at the verification and validation of the whole software system. Tests of each testing level can be performed manually or automated by means of the application of testing tools. In contrast to manual testing, *automated software testing* leads to higher effectiveness and, thus, to higher acceptance of continuously performed test execution (Dustin et al. 2009; Marcozzi et al. 2012). Therefore, the use of testing tools enables the efficient and continuous testing of the correctness of a system after changes, the so called *regression testing* (Abran et al. 2004).

Although software testing is part of the software system lifecycle, an online survey among 870 business intelligence professionals from German-speaking countries conducted in 2013 has shown that data warehouse testing is currently neglected (Krawatzeck 2013) (n = 55, response rate = 6.3%). Only 54% of the respondents said that they are currently testing their DWH in any manner. As the biggest challenge the survey identified the automation of the DWH testing process. The respondents named besides others the modest budget and the lack of appropriate tools for DWH testing as potential reasons for the omission of DWH testing in general or automated DWH testing in particular. Besides the aforementioned challenges regarding the correctness of the presented information within a DWH, the negligence of DWH testing is particularly surprising, since many DWHs have failed in the past – surveys exhibit failure rates of more than 50% (Schutte et al. 2011).

We performed a short review in order to verify the lack of appropriate tools and could identify contributions with collections of unit testing tools suitable for DWH testing (Meszaros 2007; Crispin & Gregory 2008; Collier 2011; Krawatzeck 2013). However, detailed evaluations of these tools are missing and, thus, no advice for selecting an appropriate tool is provided. The paper at hands aims at filling this gap by presenting the results of such an evaluation. To address the mentioned modest budget problem, candidates for evaluation have been limited to no charge, open source (OS) solutions.

Our evaluation contributes to the field of DWH testing in at least two ways. First, it shows that some promising OS DWH unit testing frameworks exist. Second, it can serve as a starting point and therefore reduces costs in terms of time and money for organizations planning to evaluate and select DWH testing tools (IEEE 2010). In addition, although the term 'unit testing tools' is misleading, those tools can also

be used to support and automate the integration and system testing levels. Hence, the presented work contributes to all DWH testing levels.

The remainder of the paper is organized as follows: After a short discussion on related work (Section 2) we explain the methodology we will use for the evaluation and selection of OS DWH unit testing tools (Section 3). Afterwards the detailed evaluation is presented in Section 4. Concluding, the findings are summarized and an outlook to further research is given (Section 5).

2 RELATED WORK

Within scientific literature only a few contributions address DWH testing (Golfarelli & Rizzi 2011a). Previous work mainly focuses on which test activities should be conducted (Golfarelli & Rizzi 2011a; Golfarelli & Rizzi 2011b) or which testing approach is the most suitable one for DWH testing (ElGamal et al. 2012; Gupta et al. 2012; ElGamal et al. 2013). Except for (Krawatzeck 2013), none of these papers provides any information about specific tools suitable for DWH testing.

Literature regarding the design and development of DWHs addresses the topic of testing (Inmon 1996; Kimball et al. 1998; Moss & Atre 2003), but again, does not provide any information about adequate tools. Only Meszaros (2007), Crispin and Gregory (2008), and Collier (2011) list some tools suitable for database or DWH testing, respectively. However, such lists only contain information about the tool names and download locations. Hence, no underpinned statement about the appropriateness of the tools can be made.

3 RESEARCH METHODOLOGY

Unit testing tools provide the functionality to specify and automatically execute software unit tests. Regardless whether classic or agile software development methodologies are used, testing is an important part of the software development lifecycle (ISO, IEC, IEEE 2008). Thus, unit testing tools assist software engineers within the software life-cycle process ‘verification’ and ‘validation’ and therefore belong to the class of Computer-Aided Software Engineering (CASE) tools (IEEE 2010).

The evaluation and selection process of CASE tools is well-investigated (e.g. Du Plessis 1993; Le Blanc & Korn 1994; ISO, IEC 2008; IEEE 2010). Since (IEEE 2010) is standardized and is based on the software product evaluation model as defined in (ISO, IEC 2008) and provides consistency in the evaluation and selection processes, we follow the methodology suggested by IEEE (2010) within the paper at hand.

IEEE (2010) splits the evaluation and selection of CASE tools into four major processes: (P1) preparation process, (P2) structuring process, (P3) evaluation process, and (P4) selection process. Not all of the four suggested major processes and related activities have to be passed (IEEE 2010). The evaluation and selection goal determines the selection of relevant processes and activities (IEEE 2010). Since our goal is a detailed overview of DWH unit testing tools, the final selection process (P4) is not needed and, in fact, not feasible. A unit testing tool should fit into the IT environment of an organization. Since we leave such organization specific parameters aside to provide a broad overview independently of a specific organization, a selection recommendation would be misleading. However, we perform the first three major processes and the results can serve as a starting point for evaluations conducted later by different organizations.

Within the preparation process (P1) the following activities should be performed: goal setting, establishing of selection criteria, and project planning and control. Since our project plan is not of general interest, it is not included in the paper at hand. The definition of requirements, the gathering of tool information, and the identification of final evaluation candidates are part of the structuring process (P2). The actual evaluation (P3) cuts down into preparation, application, and reporting.

Figure 1 summarizes the evaluation process including the intermediate results. Additionally, it includes the assignment of the processes and activities to the related paper sections to provide a fast overview.

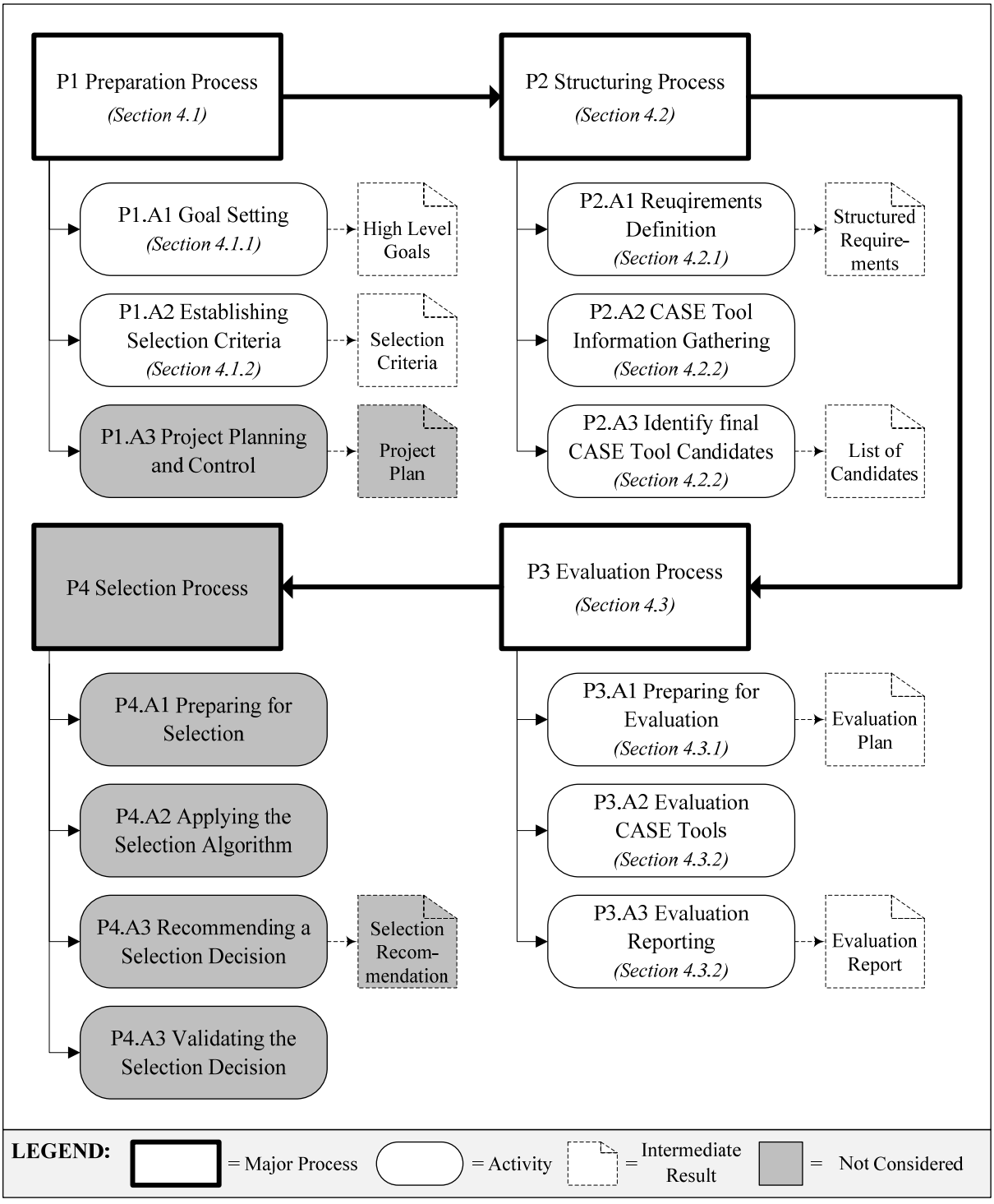

Figure 1. Evaluation and Selection Process of CASE Tools (IEEE 2010)

4 EVALUATION OF OPEN SOURCE DWH UNIT TESTING TOOLS

In the following the three major processes ‘preparation’, ‘structuring’ and ‘evaluation’ of the IEEE 14102-2010 guidelines for evaluation and selection of CASE tools are described in detail including all activities and intermediate results.

4.1 Preparation Process

The preparation process consists of the two activities ‘goal setting’ and ‘establishing selection criteria’ which are presented in the following subsections.

4.1.1 Goal Setting

Following the instructions given by IEEE (2010), we answer the questions ‘Why a DWH unit testing tool should be used?’ and ‘Of what type should it be?’ to obtain the high level goals of the evaluation. The answer to the first question sets our first high level goal (G1): Automate the verification and validation process of the DWH system lifecycle to ensure high software quality. The answer of the second question includes two high level goals. First, the unit testing tool should not charge the budget for DWH development and maintenance (G2). Second, the usage of the testing tool should not cause any security issues, since DWHs contain sensitive data as explained above (G3).

Table 2 summarizes the high level goals.

ID	Goal
G1	Automate the verification and validation process of the DWH system lifecycle to ensure high software quality
G2	Free of charge
G3	Meet security policies

Table 2. High Level Goals

4.1.2 Establishing Selection Criteria

We deduce the selection criteria from the high level goals. To meet Goal 1 we try to find a tool which is able to specify test cases through the definition of so-called test scripts (S1) and which is able to execute the test cases automatically (S2). In addition, it has to be suitable for DWH testing and therefore supports the testing of all necessary DWH test objects (S3). To meet Goal 2 the tool should be open source (S4). To avoid security issues (Goal 3), the tool should either not require database credentials at all or provide the possibility to encode them (S5).

Table 3 gives an overview about the selection criteria and describes the selection criterion S3 in more detail.

4.2 Structuring Process

The structuring process consists of the activities ‘requirements definition’, ‘CASE tool information gathering’, and ‘evaluation candidates identification’. Since we do not want to exclude any promising DWH unit testing tools a priori from our evaluation, our information gathering activity directly results in the list of evaluation candidates. Hence, the description of these two activities is merged and presented in Section 4.2.2.

ID	Selection Criteria
S1	Specify test cases
S2	Automatically execute test cases
S3	Suitable for DWH testing, i.e. it is possible to test the correctness of: • the data within the DWH, • the database schema, • stored procedures, and • views.
S4	Open source
S5	Security

Table 3. Selection Criteria

4.2.1 Requirements Definition

IEEE (2010) provides product characteristics which should be taken into account when evaluating and selecting CASE tools. These characteristics are divided into different characteristic groups related to (A) life-cycle process functionalities, (B) CASE tool usage functionalities, (C) general quality, and (D) general characteristics not related to quality. Each characteristic group is further differentiated in several sub-characteristics (IEEE 2010).

We select the following sub-characteristics (cf. Table 4) and structure them with regard to the five selection criteria (S1–S5). Since not all of the by IEEE (2010) suggested sub-characteristics are clearly defined for unit testing tools, Table 4 also contains the specific questions which we assess in our specific case.

Sub-Characteristic	Definition by IEEE (2010) // Assessment questions
Requirements Class C1: Sub-characteristics related to the CASE tool use (IEEE 2010) (S1)	
Hardware and Software Environment of Tool Products	Attributes related to the set of hardware and software items on which or with which products the tool can be used. *Question:* Which software is necessary to define and execute test cases?
Languages Supported	Attributes related to its ability to support specific languages. Divided into the following questions: *Question1 (Source Code):* In which programming language is the tool written? *Question2 (Test Case Definition):* Which (programming) language is used to define the test cases?
Databases Supported	Attributes relating to its ability to support specific databases. *Question:* Which specific DBMSs and/or generic database driver are supported?
Requirements Class C2: Sub-characteristics related to the construction activity (IEEE 2010) (S1)	
Report Generation	Attributes related to its ability to automate the development of reports to be produced by the system under development (as opposed to the CASE tool). *Question:* Is it possible to automatically generate reports about implemented test suites and test cases?
Requirements Class C3: Sub-characteristics related to the validation process (IEEE 2010) (S1 & S2)	
Test Case and Expected Result Entry	Attributes related to its ability to support user entry of test cases and entry of expected test case results.
Test Case and Expected Result Generation	Attributes related to its ability to automatically generate test cases based upon existing requirements and/or design specification data available to the tool and to automatically generate expected test case results.
Test Traceability	Attributes related to traceability of test activities and data.

Sub-Characteristic	Definition by IEEE (2010) // Assessment questions
Test Driving	Attributes related to its ability to execute and/or replay test cases.
Run-time Analysis	Attributes related to its ability to analyze the performance of a program as it executes.
Regression Testing	Attributes related to its ability to support regression testing.
Automatic Result checking	Attributes related to its ability to automatically compare expected test case results and actual test case results.
Requirements Class C4: Sub-characteristics related to the DWH testing suitability (S3)	
Test Object 'Data'	*Question:* Is it possible to test the correctness of the data contained within the DWH?
Test Object 'Database schema'	*Question:* Is it possible to test the correctness of the database schema of the DWH?
Test Object 'Stored procedures'	*Question:* Is it possible to test the correctness of the stored procedures within the DWH?
Test Object 'Views'	*Question:* Is it possible to test the correctness of the views within the DWH?
Requirements Class C5: Sub-characteristics related to the acquisition process (IEEE 2010) (S4)	
Cost of Tool Implementation	Attributes related to the cost of implementing the tool. *Question:* What is the purchase price of the tool?
Licensing Policies	Attributes related to the supplier's licensing policies. *Question:* Under which specific OS license is the tool published?
Requirements Class C6: The criterion 'Security' from general quality characteristics (IEEE 2010) (S5)	
Security	Attributes related to its ability to prevent unauthorized use or misuse of itself. *Question:* Is it unnecessary to provide database credentials or is it at least possible to encode them?

Table 4. Structured Requirements

For an additional comprehensive set of potential evaluation criteria for software testing tool evaluation, i.e. suitable for an organization tailored evaluation, we refer to (Illes et al. 2005).

4.2.2 DWH Unit Testing Tool Information Gathering and Selection of Evaluation Candidates

Starting from the lists of possible DWH unit testing tools identified in Section 2 (Meszaros 2007; Crispin & Gregory 2008; Collier 2011; Krawatzeck 2013), we first merge the lists and afterwards select the OS solutions. Resulting in: AnyDbTest (n.d.), BI.Quality (ORAYLIS n.d.), DbFit (n.d.), DbUnit (n.d.), NDbUnit, (n.d.), SQLUnit (n.d.), TSQLUnit (n.d.), and utPLSQL (n.d.).

The candidates can be classified into 'external third party tools' which have to be integrated in an additional integrated development environment (IDE), 'external standalone third party tools' and 'in-database tools' which have to be installed on the corresponding database management system (DBMS).

Table 5 summarizes the evaluation candidates in alphabetical order.

Tool Name	Tool Version	URL	Class
AnyDbTest	2.4.5	http://anydbtest.codeplex.com/	standalone
BI.Quality	2.0.1	http://biquality.codeplex.com/	external (IDE)
DbFit	2.0.0	http://benilovj.github.io/dbfit/	standalone
DbUnit	2.4.9	http://www.dbunit.org/	external (IDE)
NDbUnit	1.6.7.0	http://code.google.com/p/ndbunit/	external (IDE)
SQLUnit	1.5	http://sqlunit.sourceforge.net/	standalone
TSQLUnit	0.91	http://sourceforge.net/apps/trac/tsqlunit	in-database
utPLSQL	2.2	http://utplsql.sourceforge.net/	in-database

Table 5. List of Candidates

4.3 Evaluation Process

The evaluation process cuts down into the preparation activity and the actual evaluation resulting in the final evaluation report as described below.

4.3.1 Preparing for Evaluation

In order to provide an adequate starting point for subsequent tailored evaluations performed by organizations, we chose mainly Boolean metrics for each sub-characteristic, i.e. a characteristic is fulfilled to its full extent or not. In some cases, where a strict Boolean decision is not possible, we additionally use the third value 'partly support'. In doing so, we give organizations the opportunity to decide on their own, to which extend these characteristics have to be supported. Hence, we do not exclude possibly valuable tools.

For the assessment of each characteristic mainly the tools websites and documentations are used (cf. Table 5). Hence, our *evaluation plan* consists of the three steps: (1) checking the tool website, (2) obtaining tool documentation, and finally (3) assessment for each of the eight evaluation candidates.

4.3.2 Evaluation DWH Unit Testing Tools and Evaluation Report

The results of the evaluation are presented within Table 6. It is noticeable that the evaluated tools exhibit similar assessments regarding the requirements addressing the validation process itself (requirements class C3). All tools allow the manual specification of test cases but no tool provides functions for automated test case generation from test case or system specifications. On the contrary, test results are checked automatically by all tools.

The requirement class C1 shows that not all tools are applicable in all IT environments. The possible range depends on either which DBMS is supported by the tool or if the tool is vendor or language independent, respectively. Language-specific tools like in-database tools allow the definition of test cases in the same SQL syntax as the objects they test, e.g. TSQLUnit tests are written in TSQL and utPSQL tests are defined using PLSQL. Other tools like DbUnit and NDbUnit use well-known programming languages such as Java and .NET for test case definition. However, most testing tools specify test cases through the Extensible Markup Language (XML). DbFit represents a notable exception. DbFit test cases are defined using a wiki-like syntax. By doing so, the communication between developers and end users should improve which leads to a better testing acceptance and in conclusion to a system under test with fewer defects (DbFit n.d.).

The generation of reports containing the test results (class C2) is only partly supported by DbFit, SQLUnit, and TSQLUnit.

All tools allow the testing of the correctness of data stored within a DWH (class C4). All other DWH test objects can only be tested indirectly. Again as a notable exception, only DbFit fully supports the testing of the database schemata by providing special functions for this purpose. All other tools can fulfill this job only indirectly via tweaked SQL statements, i.e. try to insert a data record with a missing or additional attribute into a table to verify the tables' structure. No tool provides native support for database view testing.

Furthermore, it is remarkable that only three tools, namely DbFit, TSQLUnit and utPLSQL fulfill the security criterion (class C6). TSQLUnit and utPLSQL are in-database tools and hence are installed within the database under test and therefore need no additional credentials. DbFit – as an external standalone tool – is the only tool evaluated which allows the encoding of necessary database credentials.

	Sub-Characteristics	AnyDbTest	BI.Quality	DbFit	DbUnit	NDbUnit	SQLUnit	TSQLUnit	utPLSQL
C1	Hardware and Software Environment	Text-Editor	MS Visual Studio	Java	Java & IDE like Eclipse	MS Visual Studio	Text-Editor	Database under test	Database under test
	Languages Supported (Source Code)	C#	C#	Java / .NET Code	Java Code	.NET Code	Java Code	TSQL Code	PLSQL Code
	Languages Supported (Test Case Definition)	XML	XML	Wiki-Syntax	Java	.NET	XML	TSQL	PLSQL
	Databases Supported	• MS SQL Server • MySQL • Oracle Database • ProstgreSQL	• MS SQL Server • OLEDB-supported databases	• DB2 • Derby • HSQLDB • MS SQL Server • MySQL • Oracle Database • PostgreSQL	• DB2 • Derby • FrontBase • H2 • HSQLDB • Informix • InterBase • MS SQL Server • MySQL • Oracle Database • OpenBase • PostgreSQL • SAP MaxDB • Sybase	• MS SQL Server • MySQL • Oracle Database • SQLite • OLEDB-supported databases	• DB2 • HSQLDB • Informix • MS SQL Server • MySQL • Oracle Database • PostgreSQL • Sybase • databases with a JDBC-driver	• MS SQL Server	• Oracle Database
C2	Report Generation	○	○	◑	○	○	◑	◑	○
C3	Test Case and Expected Result Entry	●	●	●	●	●	●	●	●
	Test Case and Expected Result Generation	○	○	○	○	○	○	○	○
	Test Traceability	○	○	○	○	○	○	○	○
	Test Driving	●	●	●	●	●	●	●	●
	Run-time Analysis	●	●	○	◑	◑	●	◑	◑
	Regression Testing	●	●	●	●	●	●	●	●
	Automatic Result Checking	●	●	●	●	●	●	●	●

	Sub-Characteristics	AnyDbTest	BI.Quality	DbFit	DbUnit	NDbUnit	SQLUnit	TSQLUnit	utPLSQL
C4	Test Object ‘Data’	●	●	●	●	●	●	●	●
	Test Object ‘Database schema’	◑	◑	●	◑	◑	◑	◑	◑
	Test Object ‘Stored procedures’	●	○	●	○	○	●	●	●
	Test Object ‘Views’	◑	◑	◑	◑	◑	◑	◑	◑
C5	Cost of Tool Implementation	0,-	0,-	0,-	0,-	0,-	0,-	0,-	0,-
	Licensing Policies	Microsoft Public License	Microsoft Public License	GNU GPL	GNU Lesser GPL	Apache License	GNU GPL	GNU Lesser GPL	GNU GPL
C6	Security	○	○	●	○	○	○	●	●

Table 6. Evaluation Report (● = supported; ◑ = partly supported; ○ = not supported)

To provide a fast overview, the evaluation results are additionally visualized as radar charts covering all Boolean sub-characteristics (the requirements class C1 and the licensing policies as part of class C5 are excluded). For better comprehensibility, the results are visualized within three separate radar charts, each covering one of the tool classes as presented in Table 5: standalone tools (cf. Figure 7), external tools which need an IDE (cf. Figure 8), and in-database tools (cf. Figure 9).

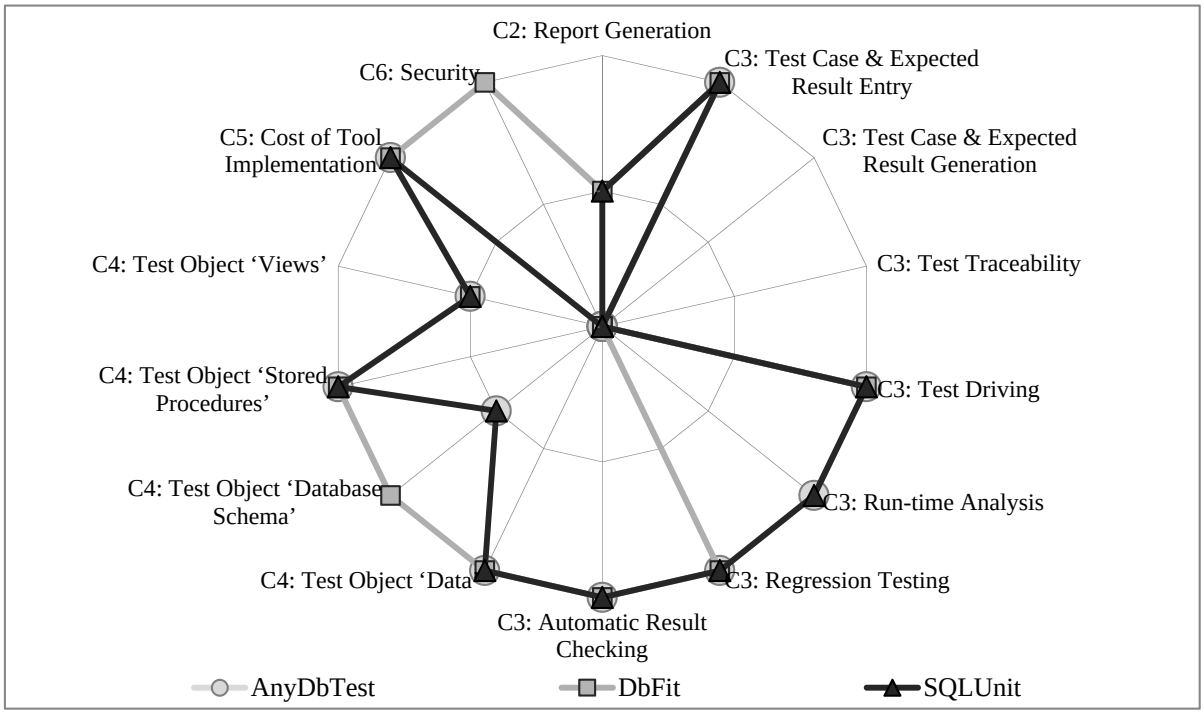

Figure 7. Evaluation Report – Tool Class 'Standalone'

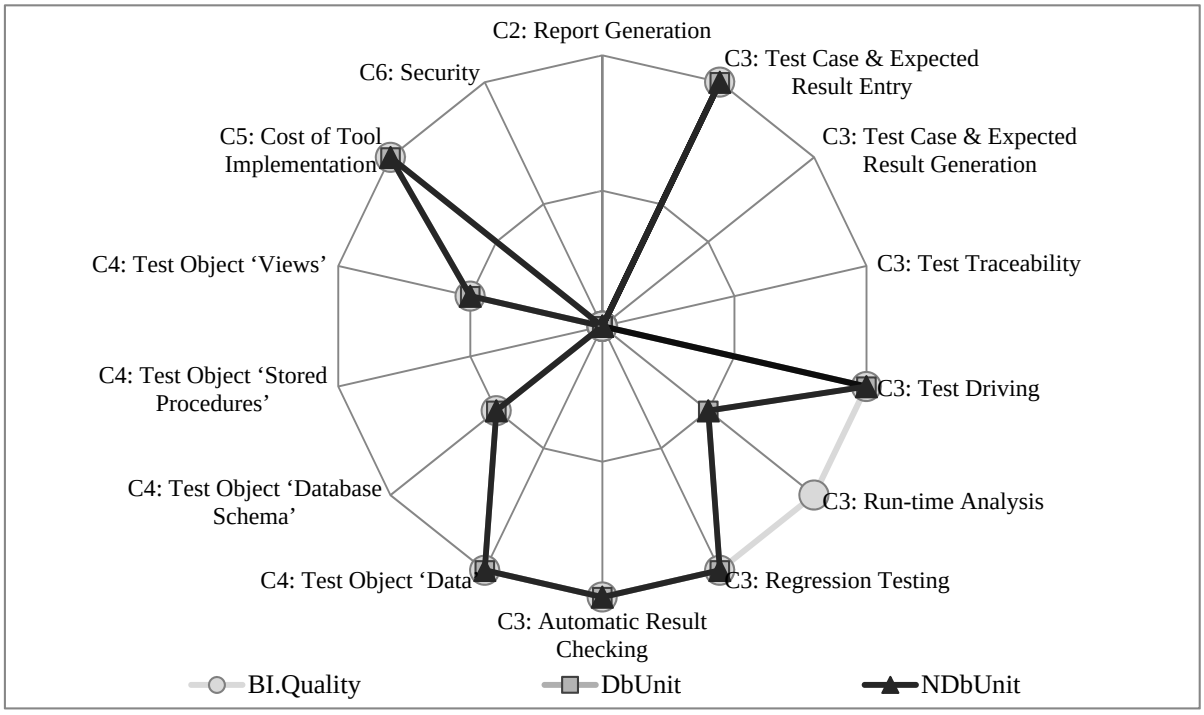

Figure 8. Evaluation Report – Tool Class 'External (IDE)'

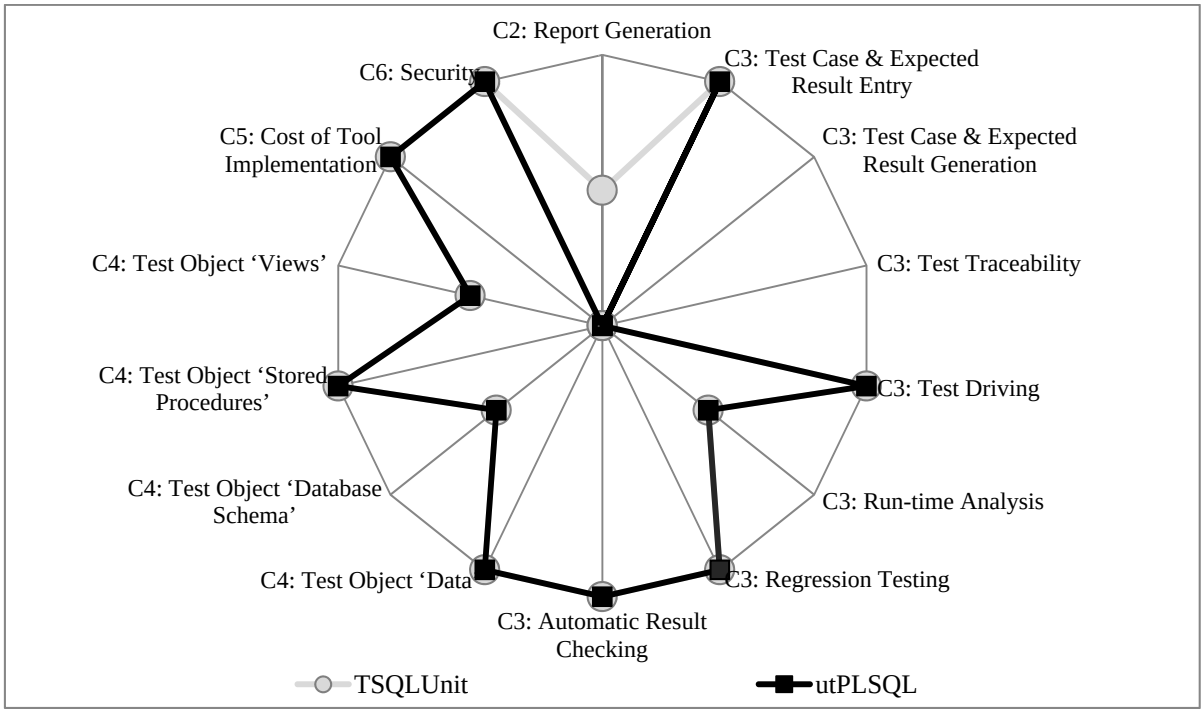

Figure 9. Evaluation Report – Tool Class 'In-database'

5 CONCLUSION AND FURTHER WORK

In the paper at hand we have motivated the importance of DWH testing and identified a gap between scientific approaches for this purpose and the actual implementation within real world scenarios. Further, we have performed an evaluation of OS DWH unit testing tools to address this issue. It has been shown, that some promising tools suitable for DWH testing exist, whereby only DbFit, TSQLUnit and utPLSQL fulfill the security requirement (no need to provide database credentials or the possibility to encode them). Since DbFit is the only DBMS vendor independent tool among these, it seems to be the most promising one. The presented results can be used as a starting point by organizations which plan to evaluate and finally select a DWH unit testing tool.

Within the support of management's decision-making processes further applications besides a DWH are involved. Analogously to DWHs these applications are subject to adaptions. Hence, additional evaluations of testing tools suitable for such application domains, like reporting, online analytical processing (OLAP), or information presentation (dashboards), should be conducted. In addition, our evaluation revealed that none of the assessed tools provides functions for automated test case generation from test case or system specifications. Since automated test case generation has the potential to further reduce costs in terms of time and money for DWH testing, further work should evaluate if such tools exist and may suggest or develop an appropriate solution, respectively.

References

Abran, A., Moore, J. W., Bourque, P., Dupuis, R. and Tripp, L. L. (Eds.) (2004). Guide to the Software Engineering Body of Knowledge (SWEBOK). IEEE, Los Alamitos.

AnyDbTest (n.d.). http://anydbtest.codeplex.com/.

Collier, K.W. (2011). Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing. Addison-Wesley, Upper Saddle River.

Crispin, L. and Gregory, J. (2008). Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley, Upper Saddle River.

DbFit (n.d.). http://benilovj.github.io/dbfit/.
DbUnit (n.d.). http://www.dbunit.org/.
Du Plessis, A. (1993). A method for CASE toll evaluation. Information & Management, 25(2), 93–102.
Dustin, E., Garrett, T. and Gauf, B. (2009). Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality. Addison-Wesley, Upper Saddle River.
ElGamal, N., Bastawissy, A. El and Galal-Edeen, G. (2012). Towards a Data Warehouse Testing Framework. In: Proceedings of the 9th International Conference on ICT and Knowledge Engineering, pp. 65–71, ICT-KE'2011, Bangkok, Thailand.
ElGamal, N., ElBastawissy, A. and Galal-Edeen, G. (2013). Data Warehouse Testing. In: Proceedings of the Joint EDBT/ICDT Workshops 2013, pp. 1–8, EDBT/ICDT'2013, Genoa, Italy.
Golfarelli, M. and Rizzi, S. (2011a). Data Warehouse Testing: A Prototype-based Methodology. Information and Software Technology, 53(11), 1183–1198.
Golfarelli, M. and Rizzi, S. (2011b). Data Warehouse Testing. International Journal of Data Warehousing and Mining, 7(2), 26–43.
Gupta, S. L., Pahwa, P. and Mathur, S. (2012). Classification of Data Warehouse Testing Approaches. International Journal of Computers & Technology, 3(3), 381–386.
Haertzen, D. (2012). The Analytical Puzzle: Profitable Data Warehousing, Business Intelligence and Analytics. Technics Publications, Westfield.
IEEE (2010). Information Technology – Guideline for the Evaluation and Selection of CASE Tools (IEEE Std 14102-2010). IEEE, New York.
Illes, T., Herrmann, A., Paech, B. and Rückert, J. (2005). Criteria for Software Testing Tool Evaluation – A Task Oriented View. In: Proceedings of the 3rd World Congress for Software Quality, Vol. 2., pp. 213–222, 3WCSQ, Munich, Germany.
Inmon, W.H. (1996). Building the Data Warehouse. 2nd Edition. Wiley, New York.
ISO, IEC (2008). Information technology – Software Product Evaluation (ISO/IEC 14595-5:1998). ISO/IEC, Geneva.
ISO, IEC, IEEE (2008). Systems and Software Engineering – Software Life Cycle Processes (ISO/IEC 12207:2008). ISO/IEC-IEEE, Geneva.
Kimball, R., Reeves, L., Ross, M. and Thornthwaite, W. (1998). The Data Warehouse Lifecycle Toolkit – Expert Methods for Designing, Developing, and Deploying Data Warehouses. Wiley, New York.
Krawatzeck, R (2013). Pilotstudie: Praktiken und Herausforderungen beim Testen von BI-Systemen. BI-Spektrum, 8(3), 10–14.
Le Blanc, L. and Korn, W. (1994). A phase approach to the evaluation and selection of CASE tools. Information and Software Technology, 36(5), 267–273.
Manjunath, T., Hegadi, R. and Ravikumar, G. (2011). Analysis of Data Quality Aspects in Data Warehouse Systems. International Journal of Computer Science and Information Technologies, 2(1), 477–485.
Marcozzi, M., Vanhoof, W. and Hainaut, J.-L. (2012). Test input generation for database programs using relational constraints. In: Proceedings of the Fifth International Workshop on Testing Database Systems, Article No. 6, DBTest'2012, Scottsdale, USA.
Meszaros, G. (2007). xUnit Test Patterns: Refactoring Test Code. Addison-Wesley, Upper Saddle River.
Moss, L.T. and Atre, S. (2003). Business Intelligence Roadmap – The Complete Project Lifecycle for Decision-Support Applications. Addison-Wesley, Boston.
NDbUnit (n.d.). http://www.code.google.com/p/ndbunit.
ORAYLIS (n.d.). BI.Quality, http://biquality.codeplex.com/documentation.
Schutte, S., Ariyachandra, T. and Frolick, M. (2011). Test-Driven Development of Data Warehouses. International Journal of Business Intelligence Research, 2(3), 64–73.
SQLUnit (n.d.). http://sqlunit.sourceforge.net/.
TSQLUnit (n.d.). http://sourceforge.net/apps/trac/tsqlunit/.
utPLSQL (n.d.). http://utplsql.sourceforge.net/.
Wang, Y., Buranawatanachoke, S., Colle, R., Dias, K., Galanis, L., Papadomanolakis, S. and Shaft, U. (2009). Real application testing with database replay. In: Proceedings of the Second International Workshop on Testing Database Systems, Article No. 8, DBTest'2009, Providence, USA.