Todo list

Setup and Implementation of an Automated Testing Pipeline for a DataOps Use Case

Bachelor's Thesis (T3201)

presented to the
Department of Computer Science

at the
Baden-Wuerttemberg
Cooperative State University
Stuttgart

by
Oliver Rudzinski

submitted on
September 7th, 2020

Project Period (CW)	25/2020 – 36/2020
Matriculation Number, Course	`5481330`, TINF17A
Training Company	Hewlett Packard Enterprise
Internship Company	DXC Technology
Project Supervisor	Dipl.-Ing. Bernd Gloss
University Supervisor	Jamshid Shokrollahi, Ph.D.

Erklärung

Declaration of Authorship

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema:

I hereby declare that I am the sole author of this bachelor's thesis on the topic:

Setup and Implementation of an Automated Testing Pipeline for a DataOps Use Case

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

and that I have not used any sources other than those listed in the bibliography and identified as references.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

I further declare that the electronically submitted version of this thesis is identical to the printed version.

_____, _____ _____

Ort Place | Datum Date | Unterschrift Signature

Abstract

Add Abstract

Contents

List of Acronyms

BI	Business Intelligence
CI/CD	Continuous Integration & Deployment
DWH	Data Warehouse
ELT	Extract-Load-Transform
ETL	Extract-Transform-Load
DAMA	Data Management Association
MBA	Market Basket Analysis
ML	Machine Learning
SPC	Statistical Process Control
VCS	Version Control System

List of Figures

List of Tables

List of Source Code Excerpts

1 Introduction

1.1 Relation to Project Environment

1.2 Project Scope

1.3 Task Definition

1.4 Chapter Overview

2 Theoretical Backgrounds

Add chapter motivation and structure

2.1 Introduction to DataOps

Economic analysis goes here...

2.1.1 Limitations of Traditional Data Analytics Development

2.1.2 Agile Shift

2.1.3 DataOps Characteristics

2.1.3.1 Dual Pipeline Methodology

2.1.3.2 Seperation of Environments

2.1.3.3 Self-Containment of Analytics Pipeline Nodes

2.2 Introduction to Testing

In general, software development relies on testing principles to holistically and objectively validate the expected performance of a piece of software. Neglecting testing in the discipline of software engineering might lead to customers and stakeholders of the solution loosing trust in its integrity and the increased occurrence of bugs and issues. In data analytics, the urge for testing is even stronger since errors in analytics outcomes might not only slow down business processes but have an enormous negative impact on data-reliant business decisions. An analytics solution might not crash and seemingly perform as expected but its resulting reports could contain errors that

cannot be identified easily. DataOps-driven solutions also fall under this category, which is why these need to be ultimately tested for both software performance and data integrity.

To provide fundamental information on the testing principles required for designing a DataOps use case solution, the following sections define the integrity requirements of data analytics solutions (Section 2.2.1), describe applicable frameworks and processes of both software and data testing, and eventually combine these by means of the DataOps methodology.

2.2.1 Integrity Requirements of Data Analytics Solutions

In order to define integrity requirements, the following section focusses on why results of data analytics processes are important, why bad data might impact these results, and what measures need to be taken into account for the solution to provide valuable analysis results.

Nowadays, organizations depend on data analytics more than ever [1]. Business Intelligence (BI) and Data Warehouse (DWH) solutions are designed to utilize data for business-required decision making [2]. While these systems are expected to generate value, many companies lose trust in their data analytics because it might be prone to unforeseeable errors [3]. This is because BI and DWH systems rely on high-quality data in order to provide representative analytics results and business insights [1]. Unfortunately, data quality issues of various sorts and manifestations lead to the systems generating false and potentially misleading reports [4][1][5]. Trying to reverse-engineer the preliminary data problems based on faulty analytics outcomes is not a valid approach. Instead, data quality is already required at its source to allow for efficient analytics and decision-making [6][7]. Data quality assessment becomes increasingly harder due to an exponential rise in overall data volume in the past [4]. Plus, complex analytics solutions are more fault-prone, which makes it harder to assess and find potential issues in their performance [8].

Addressing data quality issues needs to be a priority when performing analytics [4][9][6]. Expecting blanket existence of data quality and, thus, not addressing it inside of the data analytics solution lifecycle, is a primary mistake [1]. Instead, data quality needs to be continuously assessed in each stage of such a solution. This includes the validation of the input data, testing for potential errors during data transformation, and making sure the output data complies with its origin [8][1][5]. Moreover, when multi-stage data analytics processes are utilized, the potential sources

of errors is also multiplied, which requires individual testing of input, transformation, and output of each stage of the solution [8].

Data quality assessment needs not only to be performed for recently generated or required data, but for the entire database in use [6][4]. Historic data is generally more prone to inconsistencies than recently acquired data since the former often originate from legacy systems, where infrastructure migrations might have lead to further quality issues [10]. This is especially crucial with ML-driven applications since historical data is used for model training. If a model does not represent a specific situation correctly, its application for recent analyses will be inconclusive as well [11]. Continuous monitoring of data quality issues within the utilization of BI solutions could prove helpful for incremental improvement of outcomes [1]. Automation of the mentioned testing and validation processes increases productivity even more by reducing human error, accelerating detection, and introducing recovery measures within the conducted analyses [8][9].

All in all, data analytics solutions require tests regarding data quality at every stage where data is introduced, transformed, or processes. Furthermore, the software needs to report recognized issues and fall back into recovery protocols, if applicable. The automation of these measures is desired for better efficiency. Since data analytics solutions remain software product, the plain functionality needs to be assessed as well by means of traditional software testing and enriched with respect to data quality.

2.2.2 Software Testing

Software testing is the process of analyzing the behavior of software, primarily to detect anomalies and defects that could be categorized as a software *bug*. Software testing is conducted under controlled conditions and includes both positive and negative presets and a defined expected outcome for each testing use case [12].

Software testing consists of various methods, types, and levels. The following section covers the relevant aspects of software testing for designing a testing concept within a data analytics solution.

2.2.2.1 Testing Methods: Black Box vs. White Box Testing

Testing methods are applied to decide the degree of abstraction of a test case. In general, either the *Black Box* or the *White Box* testing method is chosen. Black box testing is applied on test cases with a high degree of abstraction. The conducted test verifies the basic functionality of the application without considering its specific

internal code logic. Therefore, creating such a test does not require deep knowledge of the specific code structure. The test literally treats the test candidate as a black box and checks if, given a specific input, a specific expected output is achieved [12, p. 65]

On the other hand, white box testing takes the internal code logic and structure into account. Its goal is to cover all possibilities of outcome, meaning to reach as many statements, condition checks and paths of the test candidate as possible. This implies that the test can only be written with good knowledge of the test candidate's source code operation. By conducting white box testing, not only expected behaviors are assessed by means of their final results. Instead, this testing method aims to confirm the internal integrity of the test candidate [12, p. 65].

In general, both white-box and black-box testing can be leveraged for any level of software testing. In practice, black-box testing is more often used on higher levels of testing, as the internal integrity is expected to be tested on the lower levels, where white-box testing is rather utilized [13, p. 26].

Introducing agility to testing is often seen as a testing method on its own [12, p. 70]. Since DataOps is conducted in an overall agile manner, non-agile methodologies of testing are not taken into consideration within this work.

2.2.2.2 Testing Levels

Levels within software testing define the magnitude in which a test case is conducted [14]. These levels range from testing individual, granular software components in a highly controlled and isolated environment to global test runs of the overall behavior of the solution in its actual deployment environment [12]

Unit Testing Unit testing is defined as the lowest level of software testing [12, p. 65]. As the name implies, the corresponding tests take individual components (i.e., *units*) from the source code and check their behavior. Unit tests are pieces of code that invoke the chosen test unit and compare its actual result with the expected outcome [14, sec. 1]. Depending on the test purpose, unit tests apply black box testing for basic functionality checks and white box testing for deeper internal functionality assessment. This requires detailed knowledge of the code under test [12]. Because of their granularity, a large number of unit tests is often required to test the software sufficiently. Thus, unit tests need to be of low complexity. They also need to be automated and repeatable, which requires a highly isolated test run environment. In general, unit test runs must perform idempotently. Each test needs to be able to

run independently and cannot rely on outcomes of previous unit test runs. The unit test should reflect the intended usage of the test candidate, which also means that it needs to be relevant for the future of the code under test [14, sec. 2].

Integration Testing Integration testing builds up on unit testing [12, p. 66]. Performing integration tests verifies that multiple combined units are working together as expected. It focusses on the testing of interfaces that connect singular components [12, p. 66] and is seen as an important counterpart to unit testing [14, sec. 3], since it covers areas that are usually not taken into account when performing isolated unit tests. Due to more dependencies outside of a controlled testing environment, the results of such tests might not be always consistent [14, sec. 3] which implies the need for more abstract testing. It is also important that integration test only provide meaningful insights on the operation of the tested software when unit tests have been conducted and passed successfully. This is also emphasized in so-called *incremental integration testing*. This process reflects the software development lifecycle by taking the addition of new features and components into account. The principle requires that the new or updated components are independently able to perform correctly before checking their integration in a higher-level testing combination [12, p. 66].

End-to-End Testing End-to-end testing is the highest level of software testing [12, p. 67] Usually, *System Testing* is described as a level between integration and end-to-end testing. Since the underlying project is presented as a fundamental proof of concept for DataOps and contains a fairly simple system architecture, system tests do not lie inside the scope of this work. Aspects of system testing are also embodied inside end-to-end testing since it aims to be performed under (close-to) deployment requirements. This creates an idealized real-life scenario which makes use of external environments. Thus, end-to-end testing is the least isolated level and serves as the final test stage before actual deployment. Usually, this includes a run-through from start to finish [12, p. 67]. Again, end-to-end testing can only provide insightful measures about the integrity of a system when the lower test levels have passed successfully.

2.2.2.3 Testing Types

There are several testing types in software testing to further categorize the purpose of tests. Testing types either classify a subset of testing processes or are applied on top of them to increase its meaningfulness.

Functional Testing Functional testing verifies that software is created in compliance with its pre-defined, functional requirements [12, p. 69]. All traditional testing levels fall under the category of functional testing since they assess the general functionality of software and do not consider specific measures of how this functionality is achieved.

Non-Functional Testing Non-functional testing is the counterpart of functional testing [12, p. 69]. It verifies that the software achieves measures defined as non-functional requirements. This includes but is not limited to load, stress, or performance testing. While these terms are often used to describe similar activities [12, p. 70], they have slightly different meanings. Load testing describes assessing the software's limitations based on heavy loads. Stress testing goes above these limits and does also take other factors of *stress* into account (e.g., repetition of similar or identical processes, etc.). Performance tests are used for the long-term analysis of performance decrease over time [12, p. 70]. Other non-functional testing types include security and user experience. Since these aspects are not covered in this work, they are not further discussed here.

Regression Testing Regression testing is one of the most common testing types [12, p. 70] and can be found on each level of software testing [15]. It aims to uncover errors with pre-existing components when other components were newly included, changed, or removed. In other words, each part of the software must still perform correctly when a part of the features changed [12, p. 70][15]. These tests are especially required when new releases are about to be deployed into production. Generally, regression tests are conducted for both corrective and progressive reasons, meaning that they can uncover current and potential future regression issues. To achieve regression testing in practice, unit tests can be written in a more abstract manner and re-run for the entire software solution, requiring older unit tests still to pass [15]. Thus, regression tests also highly benefit from automation [12].

Smoke Testing Smoke testing describes a subset of test cases on several testing levels that assesses crucial, basic functionality of software. Smoke tests not passing can be an indicator for the core features of a (piece of) software not performing properly. They also imply that deploying the failing state of the software will most likely result in unrecoverable crashes. Using smoke tests allows for quick recognition of fundamental errors before performing other, much more granular tests [16, sec. 5].

2.2.3 Data Quality Testing

The reasoning behind data quality testing is explained in the same manner as software testing: It ensures that the result of an analysis will be conducted properly and as expected as well as that problems are caught before they can cause any harm. In general, testing the data dimension of a data analytics solution ensures data quality [17, p. 2]. Other than with software testing, data quality testing does not rely on predefined and well-documented methods. Instead, the quality of the data in question needs to be purposefully defined based on its area of application [18, p. 1][19][10, pp. 116 sq.][2, p. 667]. This type of data governance provides rules [19][10, pp. 116 sq.] which can then be embedded inside the data analytics solution.

2.2.3.1 Dimensions of Data Quality

Nevertheless, there are certain dimensions of data quality that are applied on the process of defining such rules. The Data Management Association (DAMA) in the United Kingdom defined "The Six Primary Dimensions for Data Quality Assessment" in 2013 [18, pp. 7 sqq.]. This section introduces these dimensions which will be used for defining data quality measures for the practical use case at hand at a later point of this thesis.

Completeness Data completeness describes the proportion of the stored data against the potential of being one-hundred percent complete [18, p. 8][20]. It requires that there exists a definition for the *whole* completeness of the data inside the given use case [18, p. 8]. This is because an application might require certain data items and treat others as optional. An example for the lack of data completeness might be a personal data assessment form where the last name of a person is missing, even though the corresponding input field was marked required to be filled.

Uniqueness While completeness asks for (nearly) full data existence coverage, the data *uniqueness* dimension requires it not to be more than that. Uniqueness is achieved when each unique data record only exists once inside the entire database at hand [18, p. 9]. Several sources point out that duplicate data is a primary reason for misleading analyses, which means that achieving data uniqueness should be a top priority [20][2, pp. 677 sq.]. An example for the lack of data uniqueness could be an employee database table listing 520 employees, even though only 500 people work at the given company.

Timeliness The timeliness of data is the degree to which data represents the reality from the required point in time. Whether a specific data record (or series of data records) is timely or not depends on the corresponding use case and data governance definition. This also includes the time difference of the creation of the data and its actual usage [18, p. 10]. Depending on a use case, timeliness could be defined very strictly (e.g., for analyzing trends in stock trading markets) or rather loosely (e.g., changing the primary contact address of a person).

Validity Data validity describes a data item corresponding to its expected (and therefore, pre-defined) format, schema, syntax, etc. This definition should also include a range of expected or acceptable variation thresholds [18, p. 11]. Testing for data schematics is one processes which allows for definitive objective differentiation between good and bad data [19]. When a certain data item does not comply with its expected standards, it can be considered bad. Data validity provides metrics that can be included in the analytics process to efficiently rule out data quality issues without having thorough knowledge of the meaning of the data at hand [20].

Accuracy Data accuracy is the degree to which data *correctly* describes the actual object or event existing in the real world. Defining data characteristics for accuracy can often be a long-term process which requires in-depth knowledge of the represented area of application [18, p. 9]. Data accuracy goes hand-in-hand with data timeliness since outdated data can be a primary cause for bad data accuracy. Lack of accuracy might especially occur with invalid data entries that cannot be checked for [18, p. 12]. One example might lay in the different date formats in the world. A user might presume the European data format (`DD-MM-YYYY`), even though the system requires U.S. date format (`MM-DD-YYYY`). As long as the input does not exceed the expected values, a syntactically correct but inaccurate data record is created.

Consistency Data consistency describes the absence of difference when comparing multiple representations of the same real-life object against its actual definition [18, p. 13]. In other words, two occurrences of the same object do not differ if the object itself did not change between those occurrences. An example for data inconsistency might be two data records of identical purchases made by different customers where they are being charged different total amounts (discounts excluded).

These core dimensions can be enriched by taking other, more specific factors into consideration, e.g., data usability, data confidence, data value [18, pp. 13 sq.]. It

is also notable that the weighing of the respective dimensions strictly relies on the data analytics application [18, p. 5]. This means that *Timeliness* might be of great importance in one project, but could be neglected in another. Apart from that, certain dimensions *cannot* be achieved without the existence of others, and other dimensions *might* be achieved even if others are not.

2.2.3.2 Data Quality Testing in Practice

As mentioned inside the data quality dimension definitions, areas of data quality are mostly achieved when data complies with the corresponding data governance definitions. These allow for measuring the data quality [19].

By including corresponding checks into the analytics software, not valuable data can be handled accordingly. These data quality checks are then performed before, during, and after the respective analysis. These checks must be designed in the same manner that can be seen in software testing: Either (a set of) data is good and passes the checks, or it it bad and fails [17, p. 1], which then leads to analysis recovery measures, a program exit, etc. Additionally, data profiling [21] could be used to fix recognizable and correctable issues. This could especially be useful with minor data quality inconsistencies, which might only cause problems when occurring in an extensive frequency. Plus, depending on the analytics context, some issues could even be neglected [2, pp. 678 sq.]. Especially with data quality testing in multi-stage analytics pipeline environments, internal data lineage tracing is crucial in order to find out at which point the data failed its tests [20]. Log files can then be used to reverse-engineer the data quality issue [22].

In conclusion, including quality checks based on pre-defined data governance regulations, data quality issues could be recognized and handled accordingly. This could reduce the amount and severity of analytics reports resulting from bad data.

2.2.4 Test Quality Measurement and Evaluation

An important factor for testing, both for traditional software as well as data analytics solutions, is to measure the quality of the respective software and data tests [23, p. 30].

2.2.4.1 Software Testing: Test Code Coverage

A popular method for defining test quality in software development is to check how much code was executed during the run of a particular test suite [23, p. 30]. It is expected that code with a high percentage of test code coverage will be less likely to contain unexpected faults. Microsoft suggests a code coverage of 80 percent or above for production-grade solutions [23, p. 30]. On the other hand, high test coverage does not guarantee quality in code [24, p. 8]. This is because the execution of a code piece might have been invoked but the test suite did not perform any tests on this specific fragment. It is up to the developer to create tests that are not only aiming for a high coverage as the primary subject, but to consider all possible paths throughout the code under test [24, p. 9].

There exist multiple categories of coverage that can point out lack of testing in certain areas. Based on the testing level, they can give insights on a specific code file, a specific area inside the code base or the entire code base in question.

Line Coverage defines the percentage of lines that have been called and executed without failure during the run of a test suite.

Statement Coverage defines the percentage of statements that have been called and executed without failure during the run of a test suite [24, p. 2]. Other than with line coverage, a multi-line statement (e.g., definition and initialization of a large array, formatted for better readability) is considered singularly, which could improve the precision of the coverage.

Branch Coverage checks if all paths of the code under test have been executed. This is important for conditions (i.e., `if/else if/else` statements) that need to be called in different scenarios to ensure coverage on the majority of execution cases [24, p. 2].

Function/Method Coverage represents the percentage of functions or methods that have been called and executed without failure during the run of a test suite [24, p. 2]. This process does not consider the importance or length of the functions or methods but can be used as an indicator if certain areas of the code have been left out of the testing suite entirely.

2.2.5 Data Testing: Requirements Compliance

As the findings of data testing in general (cf. Section 2.2.3) suggest, measurement of data testing quality strongly depends on the data governance requirements. All in all,

it can be suggested that test cases should be designed to enforce tests inside crucial parts of the data definition, while optional or less application-areas could be treated less strictly. As with software test code coverage, a large number of data checks that inspect a large area of the data under test may not necessarily guarantee error-free data. It solely verifies that the tested areas will not make the analytics solution break or abort its processes. Moreover, the coverage is not expected to verify that analytics reports based on the allegedly clean data will be correct and insightful. The definition of high-quality data test cases is—again—up to the developer.

2.2.6 Testing in DataOps

DataOps
ines sec-

As described previously in Section **??**, DataOps contains a two-dimensional pipeline methodology. The Value Pipeline is used for the actual, real-time, production-grade analytics while the Innovation Pipeline continuously allows for feature updates of the analytics solution. The goal of DataOps testing is now to ensure correct and expected functionality of both dimensions [25, pp. 40 sqq.]. It is now to take the findings from the previous sections and to deduct an abstract, high-level framework for DataOps testing.

In general, the "Duality of testing in DataOps" [25, p. 40] now also requires two dimensions of tests which considers one of the two pipelines fixed, respectively. When testing the Value Pipeline, the underlying source code of the solution is fixed while various sorts of data are expected to flow through the pipeline. On the other hand, testing the Innovation Pipeline means checking for potential errors inside the analytics solution's update beyond its data. Here, the data is fixed (thus, pre-defined) and used to check whether the updated source code is ready to be deployed into production [25, p. 40].

2.2.6.1 Value Pipeline: Data Quality Testing

Various data flows through the Value Pipeline continuously. Traditionally, its goal is to take presumably valuable input data, perform various forms of data transformation and analysis, and report its output to some kind of knowledge base. With the findings on data quality testing (cf. Section 2.2.3), input data should not be presumed *valid*. Now, another important purpose of the Value Pipeline is to conduct applicable data tests and enable measures if incorrect data is recognized. This process needs to be applied at each point where data is ingested, integrated, transformed or derived [26][27]. During the pipeline's performance, the source code (i.e., its features and modus operandi) do not change which is why it is the data stream that needs to be

continuously verified. Enabling logging could allow for insightful operations processes finding the cause of the data problem.

2.2.6.2 Innovation Pipeline: Software Testing

The process within the Innovation Pipeline means the development and preparation of new features that are expected to improve the behavior of the analytics solution. Here, the validity of the new update needs to be validated by means of traditional software testing. In this case, using production-grade input data might not ensure the proper performance of the new solution.

This is why the input data is prepared, taking various cases of data quality (based on the use case definition) into account. This set of test data should include *bad data* which checks if the rejection and recovery measures of the new solution are in place. On the other hand, *good data* needs to be provided [28, pp. 234 sq.] which will pass all data-related checks inside the source code and move on to the actual transformation and analysis steps, which then create output data that can be used for the analysis performance evaluation. In general, it is recommended to prepare generally applicable data based on the defined data governance specification rather than picking random data (with, thus, random errors) [10, p. 115].

Plus, regression testing is a very important factor inside the Innovation Pipeline [20] since data sets generated prior to the analytics solution update still need to be processable by the new release. Therefore, in case the update is based on a data format change, historic test data should also be included in the test data set. This can be seen as a parallel to pure software regression testing where previously designed unit tests are kept to ensure that a feature update does not collide with unchanged features. However, in case of a drastic feature change, the abstraction or complete removal of tests might be reasonable, depending on the architecture and specific feature update at hand.

In conclusion, holistic testing inside a DataOps environment is achieved by taking the change of both source code and data into account. The source code should contain a number of data quality checks based on its data governance specification. This approach of data quality testing is expected to allow for confidence in the analytics solution. On the other hand, the source code needs to be tested by means of traditional software testing methods, including the validation of the data quality checks and of all transformation and analysis actions. This is to be supported by an extensive test

data repository which needs to be designed based on the production-grade data specifications. These tests are expected to invoke the majority of outcome possibilities, leading to a high test coverage and correct performance validation [29].

This testing duality concept is used for the design of the specific use case testing framework for this thesis' project in Chapter 4.3.

2.3 Technical Fundamentals

2.3.1 Microservices

2.3.2 Python Programming Language

2.3.3 Statistical Process Control (SPC)

2.3.4 Version Control System (VCS)

2.3.5 Continuous Integration & Deployment (CI/CD)

3 Analytics Pipeline DataOps Enablement

Add chapter motivation and structure

3.1 Actual State Analysis: MBA Data Analytics Pipeline

3.2 DataOps Enablement Requirements

3.3 Architecture Design

3.4 Modus Operandi

3.5 Implementation

4 DataOps Testing

Add chapter motivation and structure

4.1 Testing Strategy

4.2 Testing Architecture Design

4.3 Implementation

5 Solution Evaluation

6 Conclusion

Bibliography

[1] Munawar, N. Salim, and R. Ibrahim, "Towards Data Quality into the Data Warehouse Development", in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, 2011, pp. 1199–1206.

[2] M. Souibgui, F. Atigui, S. Zammali, S. Cherfi, and S. B. Yahia, "Data quality in ETL process: A preliminary study", *Procedia Computer Science*, vol. 159, pp. 676–687, 2019, Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 23rd International Conference KES2019, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.09.223.

[3] BI-Survey.com, *Data Quality and Master Data Management: How to Improve Your Data Quality.* [Online]. Available: https://bi-survey.com/data-quality-master-data-management (visited on 07/15/2020).

[4] J. Freudiger, S. Rane, A. E. Brito, and E. Uzun, "Privacy Preserving Data Quality Assessment for High-Fidelity Data Sharing", in *Proceedings of the 2014 ACM Workshop on Information Sharing Collaborative Security*, ser. WISCS '14, New York, NY: Association for Computing Machinery, 2014, pp. 21–29. DOI: 10.1145/2663876.2663885. [Online]. Available: 11.

[5] T. C. Redman, "Assess Whether You Have a Data Quality Problem", *Harvard Business Review*, Jul. 2016. [Online]. Available: https://hbr.org/2016/07/assess-whether-you-have-a-data-quality-problem (visited on 07/15/2020).

[6] ——, "To Improve Data Quality, Start at the Source", *Harvard Business Review*, Feb. 2020. [Online]. Available: https://hbr.org/2020/02/to-improve-data-quality-start-at-the-source (visited on 07/15/2020).

[7] Tricentis, *What is BI/Data Warehouse Testing?* [Online]. Available: https://www.tricentis.com/products/what-is-bi-data-warehouse-testing/ (visited on 07/15/2020).

[8] H. Homayouni, S. Ghosh, and I. Ray, "An Approach for Testing the Extract-Transform-Load Process in Data Warehouse Systems", in *Proceedings of the 22nd International Database Engineering Applications Symposium*, ser. IDEAS 2018, Villa San Giovanni, Italy: Association for Computing Machinery, Jun. 2018, pp. 236–245, ISBN: 9781450365277. DOI: 10.1145/3216122.3216149.

[9] M. Veber, "Automation of data quality testing", *PwC Financial Services Institute blog*, Oct. 2018. [Online]. Available: https://www.pwc.com/us/en/industries/financial-services/research-institute/blog/automation-data-quality-testing.html (visited on 07/15/2020).

[10] H. M. Sneed, B. Demuth, and B. Freitag, "A Process for Assessing Data Quality", in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, 2013, pp. 114–119.

[11] T. C. Redman, "If Your Data Is Bad, Your Machine Learning Tools Are Useless", *Harvard Business Review*, Apr. 2018. [Online]. Available: https://hbr.org/2018/04/if-your-data-is-bad-your-machine-learning-tools-are-useless (visited on 07/15/2020).

[12] A. S. Mahfuz, "Software Quality Assurance", in. Boca Raton, FL: CRC Press, 2016, ch. Testing, pp. 59–71, ISBN: 978-1-4987-3555-1. [Online]. Available: https://learning.oreilly.com/library/view/software-quality-assurance/9781498735551 (visited on 07/17/2020).

[13] P. Ammann and J. Offutt, "Introduction to Software Testing", in. Cambridge, UK: Cambridge University Press, 2017, ch. Model-Driven Test Design, pp. 19–28, ISBN: 978-1107172012. [Online]. Available: https://books.google.de/books?id=58LeDQAAQBAJ&pg=PA26&redir_esc=y#v=onepage&q&f=false.

[14] R. Osherove, "The Art of Unit Testing", in, 2nd ed. Shelter Island, NY: Manning Publications, 2013, ch. The basics of unit testing.

[15] A. P. Mathur, "Foundations of Software Testing", in, 2nd ed. Dehli, Chennai: Pearson India, 2013, ch. Test Selection, Minimization, and Prioritization for Regression Testing, ISBN: 9789332517660. [Online]. Available: https://learning.oreilly.com/library/view/foundations-of-software/9788131794760/ (visited on 07/17/2020).

[16] A. Tarlinder, "Developer Testing: Building Quality into Software", in. Crawfordsville, IN: Addison-Wesley Professional, 2016, ch. The Testing Vocabulary. [Online]. Available: https://learning.oreilly.com/library/view/developer-testing-building/9780134291109 (visited on 07/17/2020).

[17] S. Savanur and K. S. Shreedhara, "Automated data validation for data warehouse testing", in *2016 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT)*, 2016, pp. 223–226.

[18] N. Askham, D. Cook, M. Doyle, H. Fereday, M. Gibson, U. Landbeck, R. Lee, C. Maynard, G. Palmer, and J. Schwarzenbach, "Defining Data Quality Dimensions", *DAMA UK*, Oct. 2013. [Online]. Available: https://www.whitepapers.em360tech.com/wp-content/files_mf/1407250286DAMAUKDQDimensionsWhitePaperR37.pdf (visited on 05/17/2020).

[19] I. Schieferdecker, "(Open) Data Quality", in *2012 IEEE 36th Annual Computer Software and Applications Conference*, 2012, pp. 83–84.

[20] S. Shen, "7 Steps to Ensure and Sustain Data Quality", *Towards Data Science*, Jul. 2019. [Online]. Available: https://towardsdatascience.com/7-steps-to-ensure-and-sustain-data-quality-3c0040591366 (visited on 07/15/2020).

[21] *Data Profiling*, Gartner Glossary. [Online]. Available: https://www.gartner.com/en/information-technology/glossary/data-profiling (visited on 07/15/2020).

[22] N. Askham, "What do you include im Data Quality Issue Log?", *NicolaAskham.com*, Feb. 2019. [Online]. Available: https://www.nicolaaskham.com/blog/2018-21-02what-do-you-include-in-data-quality-issue-log (visited on 07/15/2020).

[23] L. Brader, H. Hilliker, and A. C. Wills, "Testing for Continuous Delivery with Visual Studio 2012", in. Microsoft, 2012, ch. Unit Testing: Testing the Inside, ISBN: 978-1-62114-019-1.

[24] S. Heckman, "Code Coverage and Static Analysis", *CSC216: Programming Concepts*, Mar. 2014, North Carolina State University. [Online]. Available: https://www.csc.ncsu.edu/courses/csc216-common/Heckman/lectures/05_Coverage_StaticAnalysis.pdf (visited on 07/20/2020).

[25] C. Bergh, G. Beghiat, and E. Strod, *The DataOps Cookbook*, 2nd ed. Cambridge, MA: DataKitchen, 2019.

[26] D. Layton, "DataOps: Building Trust in Data through Automated Testing", Apr. 2019. [Online]. Available: https://www.linkedin.com/pulse/dataops-building-trust-data-through-automated-testing-dennis-layton/ (visited on 07/20/2020).

[27] "Add DataOps Tests to Deploy with Confidence", *DataKitchen*, Apr. 2020. [Online]. Available: https://medium.com/data-ops/add-dataops-tests-to-deploy-with-confidence-4efde90869a6 (visited on 07/20/2020).

[28] J. Held and R. Lenz, "Towards Measuring Test Data Quality", in *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, ser. EDBT-ICDT '12, New York, NY: Association for Computing Machinery, Mar. 2012, pp. 233–238. DOI: 10.1145/2320765.2320830.

[29] "Add DataOps Tests for Error-Free Analytics", *DataKitchen*, Apr. 2020. [Online]. Available: https://medium.com/data-ops/add-dataops-tests-for-error-free-analytics-741ee48bd5cc (visited on 07/20/2020).