

**Masterarbeit**  
**in der Angewandten Informatik**  
Nr. AI-2021-MA-20

# **Qualitätssicherung in einem Business Intelligence-System**

**unter Verwendung automatischer Tests**

**Martina Linsel**

Abgabedatum: 15.04.2022

Prof. Dr. Ines Rossak  
Steffen Schramm

## Kurzfassung

Entscheidungen, die in Unternehmen getroffen werden, haben die Absicht, die Wirtschaftlichkeit des Unternehmens zu steigern. Diese Entscheidungen basieren auf im Unternehmen gespeicherten Daten. Die Menge der Daten, die in Unternehmen verarbeitet werden muss, wird zunehmend größer. Diese steigt stetig durch die Verwendung von ERP- und CRM-Systemen sowie weiterer externer Datenquellen. Die Speicherung dieser zahlreichen Daten erfolgt mit der Absicht, die Aussagekraft der Datenbasis zu erhöhen, um jederzeit einen Überblick über die Unternehmenszahlen zu haben und fundierte Entscheidungen zur Unternehmenssteuerung treffen zu können. Voraussetzung für diese Vorgehensweise ist die Sicherheit, jederzeit die korrekten Zahlen im dafür genutzten Business Intelligence-System vorliegen zu haben.

Um diese Sicherheit zu erhalten, muss die Qualität des Systems gewährleistet sein. In dieser Arbeit wird untersucht, ob sich die Maßnahmen zur Qualitätssicherung in OLTP-basierten Softwareprodukten auf OLAP-basierte Business Intelligence-Systeme übertragen lassen.

Zu Beginn erfolgt eine Untersuchung des möglichen Aufbaus eines Business Intelligence-Systems. Darauf folgt eine Analyse der möglichen Qualitätssicherungsmaßnahmen in OLTP-basierten Softwareprodukten und eine Betrachtung, welche dieser Maßnahmen auf Business Intelligence-Systeme anwendbar sind. Für alle Komponenten eines Business Intelligence-Systems folgen mögliche Testfälle sowie im Anschluss die Implementierung eines prototypischen Softwaretests, welcher einen Teil des ETL-Prozesses eines realen Business Intelligence-Systems testet. Abschließend wird die Frage beantwortet, ob sich die Qualitätssicherungsmaßnahmen der klassischen Softwareentwicklung auf Business Intelligence-Systeme anwenden lassen.

## **Abstract**

Decisions made in companies have the intention of increasing the profitability of the company. These decisions are based on data stored in the company. The amount of data that needs to be processed in companies is becoming increasingly larger. This is constantly increasing due to the use of ERP and CRM systems as well as other external data sources. This large amount of data is stored with the intention of increasing the informative value of the database in order to have an overview of the company figures at all times and to be able to make well-grounded decisions for company management. The prerequisite for this approach is the certainty that the correct figures are available at all times in the business intelligence system used for this purpose.

In order to obtain this security, the quality of the system must be guaranteed. This paper investigates whether the quality assurance measures in OLTP-based software products can be transferred to OLAP-based business intelligence systems.

At the beginning an investigation of the possible structure of a Business Intelligence system takes place. This is followed by an analysis of possible quality assurance measures in OLTP-based software products and a consideration of which of these measures are applicable to business intelligence systems. For all components of a Business Intelligence system possible test cases follow as well as the implementation of a prototypical software test, which tests a part of the ETL process of a real Business Intelligence system. Finally the question is answered whether the quality assurance measures of the classical software development can be applied to Business Intelligence systems.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>V</b>
<b>Tabellenverzeichnis</b>	<b>VI</b>
<b>1 Einführung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	2
<b>2 Aufbau eines BI-Systems</b>	<b>3</b>
2.1 Differenzierung von herkömmlichen Softwareprodukten und BI-Systemen . . . . .	3
2.2 Quellsysteme . . . . .	5
2.3 ETL-Prozess . . . . .	7
2.3.1 Extraktionsphase . . . . .	7
2.3.2 Transformationsphase . . . . .	8
2.3.3 Ladephase . . . . .	9
2.4 Datenspeicher . . . . .	10
2.4.1 Data Warehouse . . . . .	10
2.4.2 Data Lake . . . . .	14
2.4.3 Data Marts . . . . .	15
2.5 Analyse . . . . .	16
<b>3 Qualität in Softwareprodukten und BI-Systemen</b>	<b>18</b>
3.1 Qualität von Softwareprodukten . . . . .	18
3.1.1 Qualitätskriterien . . . . .	18
3.1.2 Sicherstellung der Qualität . . . . .	25
3.2 Qualität in BI-Systemen . . . . .	29
<b>4 Softwaretests in BI-Systemen</b>	<b>31</b>
4.1 ETL-Prozess . . . . .	31
4.2 Datenspeicher . . . . .	43
4.2.1 Data Warehouse . . . . .	43
4.2.2 Data Lake . . . . .	47
4.3 Analyse-Anwendungen . . . . .	48
<b>5 Prototypische Umsetzung geeigneter Testverfahren</b>	<b>51</b>
5.1 Aufbau des Testsystems . . . . .	51
5.2 Testfall: Slowly Changing Dimension 2 . . . . .	52
5.2.1 Ziel des Tests . . . . .	52
5.2.2 Umsetzung . . . . .	54
5.2.3 Auswertung . . . . .	57
<b>6 Zusammenfassung und Ausblick</b>	<b>59</b>

## Literaturverzeichnis

VIII

## Abbildungsverzeichnis

2.1	Genereller Aufbau einer Software nach [Hö]	4
2.2	Architektur eines BI-Systems, vereinfacht nach [eBu15, S. 5]	5
2.3	Arten der Transformation nach [KSS14, S. 28]	9
3.1	Qualitätskriterien nach McCall nach [Alk14]	21
3.2	Qualitätskriterien nach Boehm ([Gil11, S. 26] nach [BBL76])	21
3.3	Qualitätskriterien ISO/IEC 9126 [AQ] nach [ISO01]	22
3.4	Qualitätskriterien ISO/IEC 25010 nach [ISO11]	22
3.5	Korrelationsmatrix der Qualitätskriterien nach [Hof13, S. 11]	23
3.6	Qualitätssicherungsmaßnahmen nach [Hof13, S. 20]	25
5.1	Architektur des Testsystems	51
5.2	Einordnung des Testfalls in das Testsystem	54
5.3	Kennzeichnung des Testfalles als erfolgreich	57

## Tabellenverzeichnis

2.1	Slowly Changing Dimension Type 0 - Ausgangszustand . . . . .	12
2.2	Slowly Changing Dimension Type 0 - Änderung des Standort-Attributes wird nicht berücksichtigt . . . . .	12
2.3	Slowly Changing Dimension Type 1 - Ausgangszustand . . . . .	12
2.4	Slowly Changing Dimension Type 1 - Änderung des Standort-Attributes . . . . .	12
2.5	Slowly Changing Dimension Type 2 - Ausgangszustand . . . . .	12
2.6	Slowly Changing Dimension Type 2 - Historisierung eines Datensatzes nach Änderung des Standort-Attributes . . . . .	13
2.7	Slowly Changing Dimension Type 3 - Ausgangszustand . . . . .	13
2.8	Slowly Changing Dimension Type 3 - Historisierung eines Datensatzes nach Änderung des Standort-Attributes . . . . .	13
2.9	Slowly Changing Dimension Type 4 - Haupttabelle mit dem aktuellen Datensatz .	13
2.10	Slowly Changing Dimension Type 4 - Historisierungstabelle . . . . .	13
2.11	Slowly Changing Dimension Type 6 - Historisierung eines Datensatzes nach Änderung des Standort-Attributes . . . . .	14
3.1	Qualitätskriterien nach [Gil11, S. 24,32,35], [Onp] nach [ISO01], [ISO11], [MRW77], [BBL76] . . . . .	19
3.1	Qualitätskriterien nach [Gil11, S. 24,32,35], [Onp] nach [ISO01], [ISO11], [MRW77], [BBL76] . . . . .	20
3.2	Qualitätskriterien eines BI-Systems nach [LG02] . . . . .	30
4.1	Testfall 1.01: Datensätzen in eine leere Datenbank laden . . . . .	33
4.2	Testfall 1.02: Hinzufügen einer leeren Datei . . . . .	33
4.3	Testfall 1.03: Wiederholtes Hinzufügen einer bereits verarbeiteten Datei . . . . .	34
4.4	Testfall 1.04: Zeitgleiches Hinzufügen mehrerer unterschiedlicher Dateien . . . . .	35
4.5	Testfall 1.05: Hinzufügen einer fehlerhaften Datei . . . . .	36
4.6	Testfall 1.06: Hinzufügen einer Datei mit fehlerhaften Datensätzen . . . . .	36
4.7	Testfall 1.07: Hinzufügen einer Datei mit veränderter Datenstruktur . . . . .	37
4.8	Testfall 1.08: Slowly Changing Dimension Typ 0 . . . . .	38
4.9	Testfall 1.09: Slowly Changing Dimension Typ 1 . . . . .	38
4.10	Testfall 1.10: Slowly Changing Dimension Typ 2 . . . . .	39
4.11	Testfall 1.11: Slowly Changing Dimension Typ 3 . . . . .	40
4.12	Testfall 1.12: Slowly Changing Dimension Typ 4 . . . . .	40
4.13	Testfall 1.13: Slowly Changing Dimension Typ 6 . . . . .	41
4.14	Testfall 2.01: Alle Pflichtattribute enthalten einen Wert . . . . .	44
4.15	Testfall 2.02: Alle Werte entsprechen den syntaktischen Anforderungen . . . . .	44
4.16	Testfall 2.03: Grenzwerte werden korrekt berücksichtigt . . . . .	45
4.17	Testfall 2.04: Berechnete Attribute enthalten die korrekten Formeln . . . . .	45
4.18	Testfall 2.05: Datei wird korrekt in gewünschter Struktur abgelegt . . . . .	47

4.19	Testfall 2.06: Datei wird nach Speichern in der Datenbank in Archivzone abgelegt	47
4.20	Testfall 3.01: Filter sind vollständig im Bericht enthalten . . . . .	48
4.21	Testfall 3.02: Korrekte Filterung aller Werte . . . . .	49
4.22	Testfall 3.03: Darstellung aller Werte im korrekten Format . . . . .	49
4.23	Testfall 3.04: Berechnungen erfolgen auf Basis der korrekten Formeln . . . . .	49
5.1	Slowly Changing Dimension: Ausgangszustand . . . . .	53
5.2	Slowly Changing Dimension: Historisierung eines Datensatzes nach Änderung des Standort-Attributes . . . . .	53
5.3	Slowly Changing Dimension: Historisierung eines Datensatzes nach Änderung des Joblevel-Attributes . . . . .	53



# 1 Einführung

## 1.1 Motivation

Entscheidungen, die in Unternehmen getroffen werden, haben die Absicht, die Wirtschaftlichkeit des Unternehmens zu stärken. Die Grundlage, auf welcher diese Entscheidungen getroffen werden, sind Daten. Die Menge der Daten wird zunehmend größer, insbesondere seit der Verwendung von operativen Systemen wie ERP- oder CRM-Systemen. Auch das Einbeziehen externer Quellen in den Datenbestand lässt diesen kontinuierlich anwachsen. Die Speicherung dieser zahlreichen Daten erfolgt mit der Absicht, die Aussagekraft der gesamten Datenbasis zu erhöhen.

Schon in den sechziger Jahren wurden Managementinformationssysteme verwendet, um die Informationen als Entscheidungsgrundlage angemessen darzustellen. Diese Systeme scheiterten jedoch an technischen Aspekten wie dem Nichtvorhandensein grafischer Benutzeroberflächen, genügend kostengünstigem Datenspeicher oder leistungsstarken, kostengünstigen Prozessoren. Heute liegen die Probleme nicht mehr in diesem Bereich, sondern in der Überlegung, wie möglichst effizient das gesamte Potenzial dieser Analyseanwendungen und somit des Unternehmens ausgeschöpft werden kann. Bis heute hat sich die Qualität dieser Systeme stark verbessert, sodass sie als verlässliche Entscheidungshilfe gelten. [BG13, S. 12ff]

Bei den beschriebenen Systemen handelt es sich um sogenannte Business Intelligence-Systeme (BI-Systeme), welche als „integrierter, unternehmensspezifischer, IT-basierter Gesamtansatz zur betrieblichen Entscheidungsunterstützung“ ([BG13, S. 13] nach [KMU07]) definiert sind. Außerdem wird darunter „die Integration von Strategien, Prozessen und Technologien verstanden, um aus verteilten und inhomogenen Unternehmens-, Markt- und Wettbewerberdaten erfolgskritisches Wissen über Status, Potenziale und Perspektiven des Unternehmens zu erzeugen“ [BG13, S. 616] Die Verwendungsmöglichkeiten eines BI-Systems sind zahlreich [Lubb]:

- Datenbestände strukturieren
- Daten für Analysen bereitstellen
- Unternehmenskennzahlen ermitteln
- Kosten und Ressourcen ermitteln
- Geschäfts- und Produktionsprozesse analysieren
- Berichte und Statistiken bereitstellen

Als konkretes Beispiel für die Anwendung eines Business Intelligence-Systems dient die Speicherung und Verwendung von mitarbeiterspezifischen Daten. Dazu gehören etwa persönliche Daten (Name, Adresse, Geburtsdatum) sowie unternehmensbezogene Daten (Einstellungsdatum, Jobtitel, Joblevel, Standort, Teamzugehörigkeit, Unternehmenszugehörigkeit). Diese Daten sind ein wichtiger Bestandteil für das Durchführen von beispielsweise Teamcontrolling (z.B. Aufwände

des Teams, Umsätze des Teams, Lohnkosten des Teams, Anzahl Teammitglieder) , Projektcontrolling (z.B. Aufwände pro Projekt, Abrechenbarkeit des Projektes, Budget und Kosten des Projektes) und Unternehmenscontrolling (z.B. gesamte Lohnkosten, gesamter Umsatz, Mitarbeiteranzahl, durchschnittliche Betriebszugehörigkeit). Die mitarbeiterspezifischen Daten sollen in dieser Arbeit als durchgängiges Beispiel dienen, um die praktische Nutzung eines BI-Systems sowie mögliche Schwierigkeiten anschaulich zu erklären.

Ein BI-System wird generell also für analytische Prozesse verwendet - für die Planung, als Entscheidungsgrundlage und zur Unternehmenssteuerung. Um auf dieser Datenbasis fundierte Entscheidungen treffen zu können, ist eine hohe Datenqualität nötig. [KSS14, S. 265]

Ist diese gegeben, existiert jedoch noch ein weiterer Faktor, der die Qualität der Analysen und somit wichtiger Entscheidungen beeinflussen kann - das BI-System selbst. Die Verlässlichkeit sowie die korrekte Funktionsweise des BI-Systems müssen sichergestellt werden.

Business Intelligence-Systeme unterscheiden sich von klassischen Softwareprodukten insofern, dass Business Intelligence-Systeme wenig Geschäftslogik beinhalten und nur wenige Einzeloperationen ausgeführt werden, dafür aber viele umfangreiche Leseoperationen. Der Anteil des enthaltenen Codes ist geringer, dafür ist viel Ladelogik enthalten. Zusammenfassend besteht der Hauptunterschied darin, dass Business Intelligence-Systeme dem OLAP-Ansatz folgen (spezialisiert auf komplexe, zeitintensive Lesetransaktionen) und klassische Softwareprodukte dem OLTP-Ansatz (unverzögliche Realisierung einfach strukturierter Transaktionen).

Auch in der klassischen Softwareentwicklung erfolgt die Sicherung der Systemzuverlässigkeit, also auch die Ermittlung von Fehlern und daraus resultierende Abweichungen von der gewünschten Qualität. Dies geschieht zu einem großen Anteil mit Hilfe der Durchführung von Softwaretests. [Sch13, S. 6]

Diese Arbeit beschäftigt sich mit bekannten Qualitätssicherungsmaßnahmen, die in aktuellen Softwareentwicklungsprozessen angewendet werden und geht dabei vorrangig auf Softwaretests ein.

## 1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, zu erforschen, ob bzw. wie die Testansätze zur Qualitätssicherung in der Softwareentwicklung in Business Intelligence-Systemen angewendet werden können.

Zu Beginn wird der mögliche Aufbau eines Business Intelligence-Systems untersucht. Dabei wird auf die Funktionen der grundlegenden Bestandteile eingegangen sowie auf Kombinationsmöglichkeiten der einzelnen Elemente. Anschließend wird die Qualität in Softwareprodukten untersucht. Es bestehen bereits unterschiedliche Modelle zur Bewertung der Qualität, auf welche näher eingegangen wird. Darauf folgt eine Untersuchung der Möglichkeiten, die Qualität in herkömmlichen Softwareprodukten sicherzustellen. Im Fokus stehen dabei Softwaretests, auf deren unterschiedliche Varianten näher eingegangen wird. Es folgt eine Übertragung dessen auf die Qualitätssicherung in Business Intelligence-Systemen. Basierend auf diesen Erkenntnissen wird anschließend untersucht, welche Arten von Tests für welche Bestandteile des BI-Systems in Frage kommen. Dafür werden für jeden Bestandteil mögliche Testfälle erstellt. Zum Schluss erfolgt eine prototypische Implementierung eines Testfalles für ein real existierendes Business Intelligence-System. Es folgt eine Auswertung, ob der implementierte Test das gewünschte Ergebnis liefern kann. Abschließend soll die eingangs gestellte Frage beantwortet werden, ob die Qualitätssicherung durch Testen in einem BI-System mit den Testansätzen aus der Softwareentwicklung durchgeführt werden kann.

## 2 Aufbau eines BI-Systems

### 2.1 Differenzierung von herkömmlichen Softwareprodukten und BI-Systemen

Um die Besonderheiten eines Business Intelligence-Systems zu verstehen, ist es notwendig, zu Beginn zu klären, wie Software definiert wird und wie sie grundlegend aufgebaut ist. Nach [IS] ist Software definiert als „sämtliche nicht physische Bestandteile eines Computers, Computernetzwerks oder mobilen Geräts. Gemeint sind die Programme und Anwendungen (wie das Betriebssystem), die den Computer für den Nutzer funktionstüchtig machen.“

Nach [DGH03, S. 54ff] ist Software aus vier unterschiedlichen Bestandteilen zusammengesetzt:

#### 1. Komponenten

Komponenten stellen Teile der Software dar, die der Verarbeitung und Datenstrukturen des Systems entsprechen. Dies sind beispielsweise Clients, Server, Filter, Objekte oder Datenspeicher. Software ist in Komponenten unterteilt, um sie mit folgenden Eigenschaften zu entwickeln:

- Modularisierung: Zerlegung eines größeren Problems in Teilprobleme [Den91, S.212]
- Datenkapselung: Daten bzw. Implementierung vor Zugriff von außen sichern [Ste11, S.8]
- Abstraktion: Objekte sind in der Lage, Aufgaben zu erledigen, Zustände weiterzugeben oder mit anderen Objekten zu kommunizieren [Ste11, S.8]

#### 2. Verbindungen

Die Verknüpfung der einzelnen Komponenten wird durch Verbindungen hergestellt. Sie lassen sich in einfache Verbindungen (Pipes, Prozeduraufrufe, Event-Verteilung) sowie komplexe Verbindungen (Client-Server-Protokoll, SQL-Verbindung zwischen DB und Applikation) unterscheiden.

#### 3. Einschränkungen

Die Architektur der Software kann durch Einschränkungen und Anforderungen bestimmt werden. Diese bestehen beispielsweise aus der Festlegung gültiger Werte für bestimmte Eigenschaften, der erlaubten Topologie oder Desing-Elementen.

#### 4. Begründungen / Grundlegende Prinzipien

Die Entscheidungen für die Auswahl einer bestimmten Architektur oder Darstellungsform werden begründet und nachvollziehbar gemacht.

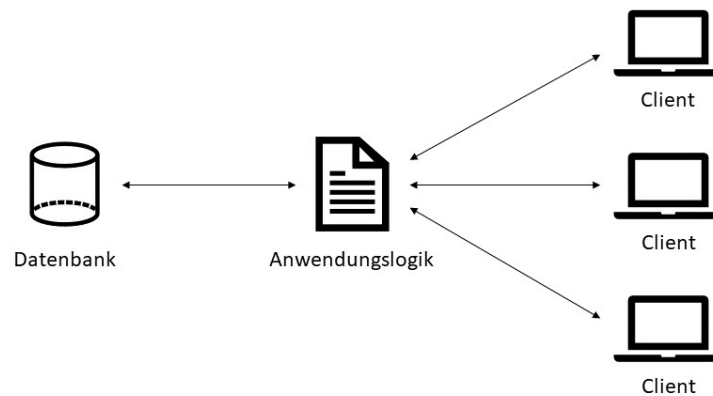


Abbildung 2.1: Genereller Aufbau einer Software nach [Hö]

In Abbildung 2.1 ist der Aufbau einer Drei-Schichten-Software-Architektur dargestellt. Enthalten sind die folgenden Komponenten:

- Datenspeicher
- Anwendungslogik (ist die eigentliche Programm-Intelligenz; verantwortlich für Koordinierung, Anforderungen und Aktualisierungen mehrerer Clients sowie Interaktionen mit dem Datenspeicher)
- Clients (stellen die visuellen Benutzeroberflächen dar; verarbeiten die Benutzereingaben)

Die Verbindungen der einzelnen Komponenten sind durch Pfeile visualisiert. [Hö]

Der grundlegende Aufbau eines BI-Systems ist in Abbildung 2.2 dargestellt. Dieses besteht aus den Komponenten Datenbank (hier Data Warehouse; andere Speicherarten sind möglich und werden in Kapitel 2.4 erläutert); weitere Datenspeicher (Quellsysteme, die sich in interne und externe Quellen unterscheiden lassen (Kapitel 2.2)) sowie Verbindungen dieser (eine besondere Bedeutung hat der ETL-Prozess, der in Kapitel 2.3 beschrieben wird). Die Komponente der Analyse (2.5) stellt dabei die Auswertung der Daten dar.

Die Datenbasis ist ein sehr wesentlicher Bestandteil des BI-Systems, da sie für alle Analysen, die in diesem System durchgeführt werden sollen, das Fundament bildet. Diese Analysen stehen im Mittelpunkt der Nutzung des Systems, weshalb die umgehende Übergabe aller relevanten Informationen an die zuständigen Nutzer im Vordergrund steht. Es handelt sich dabei um Analysen, deren Fokus auf dispositiven und strategischen Unternehmensprozessen liegt, weshalb das gesamte System an den Analyseanforderungen orientiert ist. [Hah14, S.1]

In vielen aktuellen Softwareprodukten wird für Datenbanktransaktionen OLTP (*Online Transactional Processing*) verwendet, bei welchem Datenbanktransaktionen unverzüglich realisiert werden. Dieses Vorgehen eignet sich für eine zentrale Datenhaltung, deren hauptsächliche Transaktionen einfach strukturiert sind, wie etwa das Lesen, Schreiben, Löschen und Modifizieren von einem Datensatz. Die parallele Nutzung durch sehr viele Nutzer ist möglich, wobei der Zugriff durch direkte Ein- und Ausgabe in einem Datenbankmanagementsystem oder über eine weitere Applikationssoftware geschehen kann. Im Gegensatz dazu werden im Bereich der Business

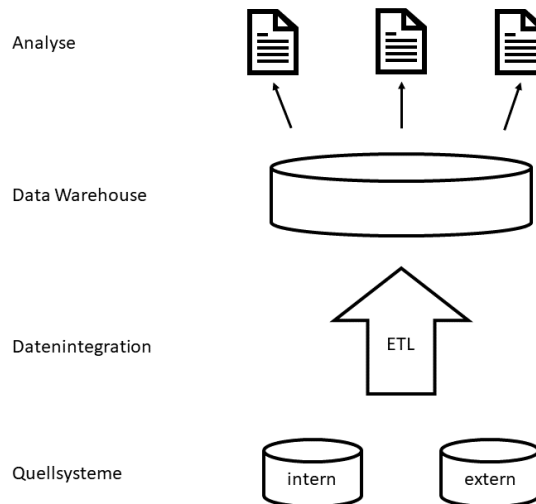


Abbildung 2.2: Architektur eines BI-Systems, vereinfacht nach [eBu15, S. 5]

Intelligence-Systeme Datenbanktransaktionen mit OLAP (*Online Analytical Processing*) realisiert, welches auf komplexe, zeitintensive Lesetransaktionen spezialisiert ist. Die Nutzung erfolgt durch nur wenige Nutzer zur gleichen Zeit, welche beispielsweise Manager, Controller oder Analysten sind. Durch diese werden unterschiedliche Transaktionen mit vielen Datensätzen auf einmal durchgeführt. Die Daten stammen aus unterschiedlichen, heterogenen Quellen. [KSS14, S. 4ff]

## 2.2 Quellsysteme

Um Analysen in BI-Systemen durchführen zu können, werden Daten benötigt. Diese werden aus sogenannten Quellsystemen abgerufen. [KSS14, S. 23] Damit bilden die Quellsysteme die Basis für den Datenfluss im gesamten BI-System, deren Qualität sich somit unmittelbar auf die Qualität der Analyseergebnisse auswirkt. [BG13, S. 45] Da für die verschiedenen Analysen eine Vielzahl an unterschiedlichen Daten benötigt wird, wurden Kriterien entwickelt, nach welchen sich die Daten klassifizieren lassen. Diese Klassifikation dient einer besseren Übersicht über das System und bringt eine verbesserte Möglichkeit, Schwierigkeiten bei der Verwendung der Daten, wie etwa unterschiedliche Sprachen oder Vertraulichkeiten, zu erkennen. Nach [BG13, S. 52ff] existieren folgende Kriterien, mit deren Hilfe Daten klassifiziert werden können:

- **Herkunft**  
Die Herkunft der Daten lässt sich in den Ursprung aus internen (z.B. ERP-, CRM- oder SCM-Systeme) und externen (z.B. Produktkataloge, Wetterdaten, Börsenkurse aus dem Internet) Quellsystemen unterteilen. [Sof] [KSS14, S. 23]
- **Zeit**  
Die Daten können aktuell oder historisiert sein.
- **Nutzungsebene**  
Es kann sich um Primärdaten oder Metadaten handeln.

- **Inhalt**  
Mögliche Formate sind z.B. Zahlen, Maßeinheiten, Zeit- oder Datumsangaben, Fotos oder Videos.
- **Darstellung**  
Die Darstellung kann z.B. numerisch oder in strukturierten Texten wie HTML oder XML erfolgen.
- **Sprache und Zeichensatz**  
Je nach Sprache können unterschiedliche Zeichensätze eingesetzt werden, wie beispielsweise der westeuropäische, griechische oder kyrillische Zeichensatz.
- **Technischer Zeichensatz**  
Es kann One-Byte-Code, Double-Byte-Code, Three-Byte-Code oder Four-Byte-Code verwendet werden.
- **Schreiborientierung**  
Je nach Sprache kann die Schriftorientierung von links nach rechts, bidirektional oder vertikal sein.
- **Schutzwürdigkeit**  
Der Grad, in dem Daten geschützt werden müssen, kann zwischen nicht erforderlichlichem Schutz, vertraulichen, geheimen oder streng geheimen Daten variieren.

Aus den für die Datenbeschaffung herangezogenen Quellsystemen muss ein Ausschnitt von Daten ermittelt werden, die im BI-System gespeichert werden sollen. [KSS14, S. 27] Nach [BG13, S. 46ff] sind wichtige Faktoren für diese Entscheidung der Zweck des BI-Systems, die Qualität der Quelldaten, die Verfügbarkeit der Daten sowie der Preis für den Erwerb der Daten.

Der *Zweck* des BI-Systems ist insofern von Bedeutung, dass die ausgewählten Daten zur Durchführung einer Analyse geeignet sein müssen. So muss beispielsweise die Aufzeichnung der Daten dem Zweck entsprechend häufig geschehen.

Die *Qualität* der Daten ist ein wichtiges Kriterium, da die Verwendung von Daten mit unzureichender Qualität für hohe Kosten oder gar ein Scheitern des Projektes verantwortlich sein kann. Die Behebung dieser qualitativen Mängel erfordert hohen Zeit- und somit auch Geldaufwand; ebenso die Korrektur von Fehlentscheidungen, die auf Basis unzureichend aussagekräftiger Analysen durchgeführt wurden. Die Arbeit mit einem solchen unverlässlichen System sorgt zudem für Unzufriedenheit unter den Nutzern. Mangelnde Qualität der Daten zeigt sich beispielsweise in inkorrekten, widersprüchlichen, unvollständigen, ungenauen oder veralteten Daten oder durch das Vorhandensein von Duplikaten.

Die *Verfügbarkeit* der Daten ist aus organisatorischer und technischer Sicht zu beachten. Auf der organisatorischen Seite muss die Verwendung der Daten z.B. durch den Betriebsrat genehmigt und der Datenschutzbeauftragte informiert worden sein. Außerdem muss beachtet werden, ob die Vertraulichkeit der Daten weiterhin gegeben ist. Die technischen Aspekte umfassen die technischen Möglichkeiten des Datenzugriffs, den Schutz vor unberechtigtem Zugriff sowie die Geschwindigkeit der Datenübertragung.

Der *Preis* der Daten ist vorallem für Daten aus externen Quellen relevant, da die Bezugsmöglichkeiten von kostenlosen Angeboten (z.B. aus dem Internet) bis zu hochpreisigen Datenquellen (z.B. Marktforschungsinstitute, Statistisches Bundesamt) reichen. Interne Quellen sind in der Regel nicht mit weiteren Kosten für die Datenbeschaffung verbunden, da diese sich im Eigentum des Unternehmens befinden.

## 2.3 ETL-Prozess

Die Abkürzung ETL bedeutet *Extraktion, Transformation, Laden* und dient dazu, Daten aus den Quellsystemen zu extrahieren, anschließend in das für Analysen benötigte Format zu transformieren und zum Schluss an den vorgesehenen endgültigen Speicherort zu laden. [KSS14, S. 97]

### 2.3.1 Extraktionsphase

Der Extraktionsprozess dient dazu, die für die geplanten Analysen benötigten Daten aus den Quellsystemen zu laden. Diese werden zu Beginn im sogenannten *Datenbereinigungsbereich*, auch *Staging Area* genannt, temporär zwischengespeichert. An dieser Stelle erfolgt anschließend die Integration und Bereinigung der Daten im Transformationsprozess. [KSS14, S. 26f]

Bevor der Extraktionsprozess beginnen kann, ist festzulegen, zu welchem Zeitpunkt die Extraktionen stattfinden sollen. Die Möglichkeiten der Extraktionszeitpunkte sind nach [BG13, S. 94] in vier Optionen gegliedert:

- **Periodisch**  
Die Extraktionen erfolgen in regelmäßigen Abständen, die von der benötigten Aktualität der Daten abhängen. Diese reicht von täglich zu aktualisierenden Daten (z.B. Arbeitszeiten der Mitarbeiter) bis hin zu selten benötigten Aktualisierungen (z.B. Standortwechsel eines Mitarbeiters).
- **Ereignisgesteuert**  
Die Extraktion erfolgt durch ein bestimmtes Ereignis, wie dem Ablaufen einer festgelegten Zeit, dem Erreichen einer bestimmten Anzahl an Transaktionen in der Datenbank oder durch ein sonstiges zu überwachendes, externes Ereignis.
- **Sofortige Extraktion**  
Bei jeder Änderung im Quellsystem wird eine Extraktion ausgelöst.
- **Anfragegesteuert**  
Die Extraktion erfolgt, nachdem ein Anwender des BI-Systems diese angefordert hat (z.B. nach Änderungen im Quellsystem, die durch den Anwender vorgenommen wurden). Diese Art der Extraktion kann ergänzend zu den vorherigen genannten Arten durchgeführt werden.

Erfolgt der Extraktionsprozess für das jeweilige Quellsystem erstmalig, müssen die gesamten Datensätze extrahiert werden. Später sind nur noch Datensätze zu extrahieren, die seit der letzten Extraktion hinzugefügt oder modifiziert wurden. [BG13, S. 111] Um Änderungen in Datenquellen erkennen und entsprechend mit einem Extraktionsprozess darauf reagieren zu können, werden Monitore verwendet. [BG13, S. 33] Wie diese funktionieren, hängt von der überwachten Datenquelle sowie den Anforderungen an die Auswertung dieser ab, weshalb typischerweise ein Monitor pro Datenquelle verwendet wird. Nach [BG13, S. 54ff] können folgende Möglichkeiten der Quellsystem-Überwachung per Monitor unterschieden werden:

- **Triggerbasiert**  
Nach jeder Transaktion der Datenbank wird der geänderte Datensatz in eine extra Datei oder Datenstruktur übertragen. Während der Extraktion müssen diese Datensätze aus der Datei ausgelesen werden.
- **Replikationsbasiert**  
Über den Replikationsdienst eines Datenbankmanagementsystems können geänderte Datensätze in eine extra Tabelle geschrieben werden. Diese Datensätze werden während der Extraktion über SQL-Abfragen aus dieser Tabelle wieder ausgelesen.
- **Zeitstempelbasiert**  
Jeder Datensatz erhält bei der Speicherung einen Zeitstempel. Wird dieser Datensatz modifiziert, erhält er einen weiteren Zeitstempel mit dem Änderungsdatum. Extrahiert werden alle Datensätze, deren Zeitstempel nach dem Datum der letzten Extraktion liegt.
- **Log-Basiert**  
Über das Datenbankmanagementsystem werden alle Transaktionen in einer Log-Datei erfasst, mit deren Hilfe alle modifizierten Datensätze ermittelt und extrahiert werden.
- **Snapshot-Basiert**  
In regelmäßigen Abständen werden die Daten aus der Datenbank in einer weiteren Datei abgelegt. Bei einem Vergleich der Datensätze ist erkennbar, welche modifiziert wurden und somit extrahiert werden müssen.

### 2.3.2 Transformationsphase

Der Transformationsprozess bringt die Daten, die im Extraktionsprozess in den Datenbereinigungsbereich geladen wurden, auf struktureller sowie inhaltlicher Ebene in ein einheitliches Format. Die Transformationen erfolgen direkt im Datenbereinigungsbereich. Erst nachdem alle notwendigen Transformationen vorgenommen wurden, werden die Daten aus dem Datenbereinigungsbereich entfernt und am nächsten Speicherort abgelegt. [BG13, S. 55] Dieser ist abhängig von der gewählten Architektur des BI-Systems.

Nach [KSS14, S. 27f] lassen sich die Transformationen in drei Bereiche einteilen, die in Abbildung 2.3 dargestellt sind.

Die erste Kategorie der Transformationen ist die *Homogenisierung* von bisher heterogenen Daten, was bedeutet, die Daten in ein einheitliches Format zu bringen, damit sie anschließend integriert werden können. Mögliche Transformationsmaßnahmen sind hier:

- Datentypen anpassen
- Zeit- und Datumsangaben vereinheitlichen
- Maßeinheiten umrechnen [BG13, S. 57]
- Attribute kombinieren oder separieren [BG13, S. 57]
- Kodierungen (z.B. von Bundesländern, Steuerklassen)



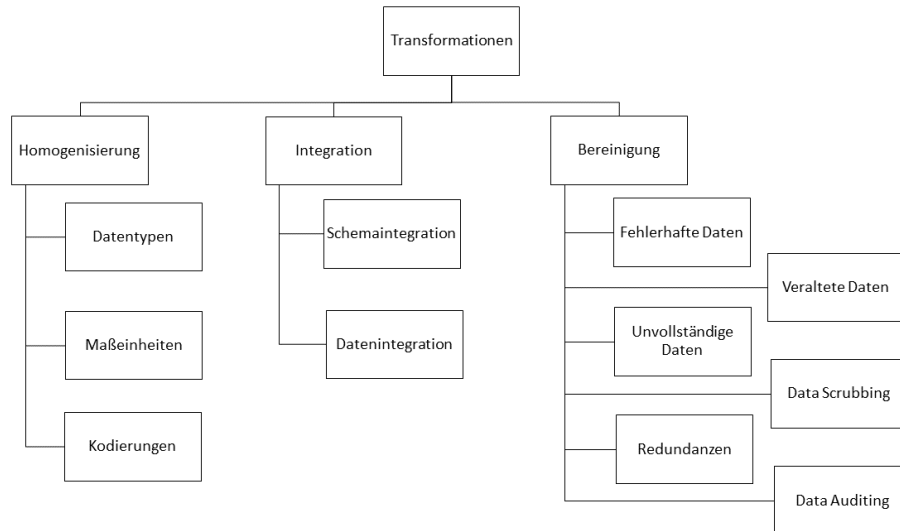


Abbildung 2.3: Arten der Transformation nach [KSS14, S. 28]

Die *Integration* ist die Zusammenführung von Daten aus verschiedenen Quellen. Dabei gibt es die Möglichkeit der Schemaintegration sowie der Instanzintegration. Bei der Schemaintegration werden neue Tabellen erstellt, die alle Datensätze der verschiedenen Quellen beinhalten. Bei der Instanzintegration erfolgt die Verbindung unterschiedlicher Datensätze.

Die *Bereinigung* der Daten dient der Verbesserung der Datenqualität. Mögliche Maßnahmen sind hier:

- Fehlerhafte Daten erkennen und reparieren
- Unvollständige Datensätze erkennen und vervollständigen
- Redundanten Datensätze (Duplikate) erkennen und bereinigen
- Veraltete Daten erkennen und aktualisieren
- Data Scrubbing (Verunreinigungen der Daten durch domänenspezifisches Wissen erkennen)
- Data Auditing (Data-Mining-Methoden verwenden, um Anomalien aufzuspüren)

Am Ende dieses gesamten Prozesses befinden sich im Arbeitsbereich nur noch Daten, die der gewünschten Datenstruktur und den Anforderungen der Analysen entsprechen. [BG13, S. 58]

### 2.3.3 Ladephase

Sind in der vorherigen Transformationsphase alle benötigten Transformationen durchgeführt wurden, werden in der darauffolgenden Ladephase alle Daten in den nächsten Speicherort transferiert. [KSS14, S. 29] Welcher das ist, hängt von der gewählten Architektur des BI-Systems ab. Wird ein Data Warehouse (siehe Kapitel 2.4.1) verwendet, werden die Daten vom Datenbereinigungsbereich in die Basisdatenbank und anschließend in die Ableitungsdatenbank geladen. Beachtet werden muss dabei eine eventuelle Historisierung, bei welcher geänderte Datensätze nicht einfach überschrieben werden dürfen, sondern zusätzlich abgelegt werden müssen. Wird ein Data

Lake (siehe Kapitel 2.4.2) verwendet, kann dieser der folgende Speicherort sein. Auch bei der Verwendung von Data Marts (siehe Kapitel 2.4.3) wird der Ladeprozess verwendet, um diese mit Daten zu befüllen. [BG13, S. 111]

Der Ladevorgang kann online oder offline geschehen. Bei einem Online-Ladevorgang ist die jeweilige Datenbank während des gesamten Prozesses weiterhin erreichbar. Im Gegensatz dazu sind die verwendeten Datenbanken während eines Offline-Ladevorganges nicht benutzbar. [BG13, S. 111] nach [AM97]

## 2.4 Datenspeicher

### 2.4.1 Data Warehouse

Der Begriff des Data Warehouses geht auf Dr. Barry Devlin zurück, welcher 1988 die erste Data Warehouse-Architektur entwickelte. [Dev97] Im Jahre 1990 definierte Bill Inmon ein Data Warehouse als subjektorientierte, integrierte, nicht flüchtige und zeitlich veränderliche Sammlung von Daten zur Unterstützung von Managemententscheidungen.

Die *Subjektorientierung* beschreibt den Hauptzweck des Data Warehouses: die Unterstützung des Anwenders beim treffen von Entscheidungen. *Integriert* verdeutlicht, dass die enthaltenen Daten aus unterschiedlichen Quellen stammen und vorhandene Inkonsistenzen möglichst gering hält. Dass die Daten *nicht flüchtig* sind, meint die Möglichkeit, eine Geschäftssituation zu einem bestimmten Zeitpunkt in der Vergangenheit wiederherzustellen. Dafür ist es nötig, nicht nur die aktuellen Daten zu speichern, sondern auch die veralteten. Diese Anforderung ist ausschlaggebend für die *zeitliche Veränderbarkeit* und bedeutet, dass jeder enthaltene Datensatz, für welchen auch die nun veralteten Datensätze gespeichert werden, einen Zeitstempel tragen muss. Die Art des Stempels ist dabei abhängig von der Art der Verwendung der veralteten Daten. [Dev18]

Nach der heutigen Definition stellt das Data Warehouse die Gesamtheit aller Daten aus allen Quellsystemen dar, die im BI-System gespeichert und für Analysen verwendet werden. Diese Daten sind strukturiert und in Dateien und Ordnern sortiert abgelegt. Es besteht aus zwei verschiedenen Datenbanken - der Basisdatenbank und der Ableitungsdatenbank. [Hah14, S. 3], [BG13, S. 58ff; S. 64ff], [KSS14, S. 31]

Die Basisdatenbank, auch *konsolidierte Datenbank*, *Datendrehscheibe* oder *operative Datenbasis* genannt, stellt die Datenbasis dar, welche sämtliche integrierte und bereinigte Daten enthält, die für spätere Auswertungen benötigt werden (Sammel- und Integrationsfunktion). Sie sind an dieser Stelle noch nicht auf bestimmte Analysen ausgerichtet, sondern anwendungsneutral abgelegt. Diese Daten haben den gesamten ETL-Prozess durchlaufen und entsprechen somit den Qualitätsstandards, die im BI-System gewünscht sind (Qualitätssicherungsfunktion). Neben den aktuellen Daten sind auch die historisierten Daten enthalten. Von dieser Datenbank aus erfolgt die Verteilung der Daten auf andere Speicherorte, was es zudem erlaubt, die Daten mehrmals zu speichern und somit mehrfach zu verwenden. Zur Wahrung der Konsistenz muss die Verteilung der Daten zwingend von dieser Datenbank ausgehen. Würde die Speicherung der Daten direkt an anderen Stellen geschehen, ohne vorher den Transformationsprozess durchlaufen zu haben und in die Basisdatenbank integriert worden zu sein, wäre die benötigte Datenqualität nicht gewährleistet (Distributionsfunktion).

Die Ableitungsdatenbank dient als Datengrundlage für alle durchzuführenden Analysen. Die enthaltenen Daten stammen aus der Basisdatenbank. Dabei transformieren die Daten vom flexiblen Datenmodell in ein auswertungsorientiertes. Sie werden dauerhaft verwaltet und bei Bedarf in der benötigten Form zur Verfügung gestellt. Auf dieser Funktionsgrundlage baut die Strukturierung der Datenbank anhand der Auswertungsbedürfnisse des Anwenders auf. Es werden einheitliche Kennzahlen und benötigte Dimensionen gebildet. Der Detaillierungsgrad der Daten kann gegenüber der Basisdatenbank abweichen. Die Verwendung kann ohne weitere Verarbeitung erfolgen, da die Daten bereits auf die geplanten Analysezwecke ausgerichtet sind. [BG13, S. 59ff] [Ale]

Die Übertragung der Daten von der Basisdatenbank in die Ableitungsdatenbank wird als Aktualisierung der Ableitungsdatenbank bezeichnet. Erfolgen regelmäßige Aktualisierungen, kann einer Inkonsistenz der Daten vorgebeugt werden. Für die Häufigkeit der Aktualisierungen gibt es drei Einteilungen [BG13, S. 58ff] [Ale]:

- **Echtzeit**  
Sobald eine Änderung im Quellsystem durchgeführt wird, wird der Prozess zur Übertragung in die Basisdatenbank angestoßen. Dafür ist eine durchgehende Verbindung der Basisdatenbank mit dem Quellsystem nötig.
- **Periodische Zeitabstände**  
Die Aktualisierung erfolgt automatisch nach dem Ablauf einer festgelegten Zeit.
- **Änderungsquantität**  
Die Aktualisierung erfolgt nach einer festgelegten Anzahl an Datenbanktransaktionen.

Bei der Übertragung der Datensätze in die Ableitungsdatenbank ist die Effizienz von großer Bedeutung. Die Durchführung über die gewöhnlichen Schnittstellen der Datenbank ist möglich, kann jedoch durch die Verwendung eines sogenannten „Bulk Loaders“ effizienter gestaltet werden. Dabei wird eine möglichst große Anzahl an Datensätzen in einer bestimmten Zeiteinheit geladen. Einige Funktionen des Datenbanksystems, wie zum Beispiel die Durchführung von Konsistenzprüfungen, sind während des Ladeprozesses nicht nutzbar, um die Performanz dessen zu erhöhen. Funktionen, die während der Verwendung eines Bulk Loaders nicht verwendbar sind, müssen somit an anderen Zeitpunkten während des ETL-Prozesses durchgeführt werden, wie etwa die Konsistenzprüfung während der Transformierung. [BG13, S. 65]

Die Arten von Daten, die in einem Data Warehouse gespeichert werden, sind unterteilbar in Fakten und Dimensionen, welche jeweils in Fakten- und Dimensionstabellen gespeichert werden. Fakten sind Bewegungstabellen aus Quellsystemen, wie etwa von Mitarbeitern durchgeführte Buchungen der Arbeitszeiten oder Absatzzahlen. Diese Art von Daten ist in großen Mengen im BI-System vorhanden und wird häufig aktualisiert. Dimensionen sind Daten, die Eigenschaften eines bestimmten Objektes enthalten, wie etwa einem Mitarbeiter (Name, Geburtsdatum, Standort, etc.). [AG]

Werden Daten in die Dimensionstabelle eines Data Warehouses hinzugefügt, bedeutet das nicht zwingend das Anlegen eines neuen Datensatzes. Existiert bereits ein Datensatz für das betreffende Objekt, gibt es verschiedene Möglichkeiten für weitere Vorgehensweisen, die sogenannten *Slowly Changing Dimensions (SCD)*. Welche Art der SCD verwendet wird, kann für jedes Attribut individuell festgelegt werden. Möglich ist beispielsweise das Übernehmen einer Änderung in den bestehenden Datensatz nach einer Änderung des Geburtsdatums eines Mitarbeiters (SCD Type 1) oder das Anlegen eines neuen Datensatzes nach der Änderung des Standortes eines

Mitarbeiters (SCD Type 2). [Dat]

*Slowly Changing Dimension Type 0:* Die bisher vorhandenen Daten können erhalten bleiben und die aktuellen Daten werden nicht in die Datenbank aufgenommen.

Tabelle 2.1: Slowly Changing Dimension Type 0 - Ausgangszustand

EmplID	Name	Standort
emp_1	Stefan Hilbrecht	Hamburg

Tabelle 2.2: Slowly Changing Dimension Type 0 - Änderung des Standort-Attributes wird nicht berücksichtigt

EmplID	Name	Standort
emp_1	Stefan Hilbrecht	Hamburg

*Slowly Changing Dimension Type 1:* Die bisher vorhandenen Daten werden mit den aktuellen Daten überschrieben. Die bisherigen Daten sind anschließend nicht mehr in der Datenbank vorhanden.

Tabelle 2.3: Slowly Changing Dimension Type 1 - Ausgangszustand

EmplID	Name	Standort
emp_1	Stefan Hilbrecht	Hamburg

Tabelle 2.4: Slowly Changing Dimension Type 1 - Änderung des Standort-Attributes

EmplID	Name	Standort
emp_1	Stefan Hilbrecht	Berlin

*Slowly Changing Dimension Type 2:* Die bisherigen Daten bleiben in der Datenbank erhalten und die aktuellen Daten werden in einem neu erstellten Datensatz abgelegt. Beide Datensätze erhalten Gültigkeitsdaten und eine Kennzeichnung, welcher Datensatz der aktuell gültige ist (textit1 bedeutet gültig, 0 bedeutet ungültig). Tabelle 2.5 zeigt den Datensatz, der die bisherigen Daten enthält. In Tabelle 2.6 ist erkennbar, dass die aktuellen Daten in einem neuen Datensatz hinzugefügt wurden und das Attribut des Standortes aktualisiert wurde. Der Erhalt der bisherigen Daten wird als *Historisierung* bezeichnet.

Tabelle 2.5: Slowly Changing Dimension Type 2 - Ausgangszustand

EmplID	Name	Standort	GuelteigAb	GuelteigBis	Guelteig
emp_1	Stefan Hilbrecht	Hamburg	01.01.1900	31.12.2999	1

*Slowly Changing Dimension Type 3:* Bei dieser Art der Historisierung existieren mehrere Spalten für ein verändertes Attribut. In dem bisherigen Datensatz ist ein Attribut enthalten, welches vorher noch nicht geändert wurde. Dieser Wert steht in der Spalte für den aktuellen Wert des

Tabelle 2.6: Slowly Changing Dimension Type 2 - Historisierung eines Datensatzes nach Änderung des Standort-Attributes

EmpID	Name	Standort	GueltigAb	GueltigBis	Gueltig
emp_1	Stefan Hilbrecht	Hamburg	01.01.1900	31.01.2022	0
emp_1	Stefan Hilbrecht	Berlin	01.02.2022	31.12.2999	1

Attributes sowie in der Spalte für den bisherigen Wert des Attributes (Tabelle 2.7). Nach der Aktualisierung der Daten wird die Spalte für den Wert des aktuellen Attributes zu dem neuen Wert geändert (Tabelle 2.8).

Tabelle 2.7: Slowly Changing Dimension Type 3 - Ausgangszustand

EmpID	Name	Aktueller Standort	Bisheriger Standort
emp_1	Stefan Hilbrecht	Hamburg	Hamburg

Tabelle 2.8: Slowly Changing Dimension Type 3 - Historisierung eines Datensatzes nach Änderung des Standort-Attributes

EmpID	Name	Aktueller Standort	Bisheriger Standort
emp_1	Stefan Hilbrecht	Berlin	Hamburg

*Slowly Changing Dimension Type 4:* Für die Speicherung der bisherigen Daten wird eine zusätzliche Tabelle angelegt (Tabelle 2.8). In der Haupttabelle ist jeweils nur der neuste Datensatz enthalten (Tabelle 2.9).

Tabelle 2.9: Slowly Changing Dimension Type 4 - Haupttabelle mit dem aktuellen Datensatz

EmpID	Name	Standort
emp_1	Stefan Hilbrecht	Hamburg

Tabelle 2.10: Slowly Changing Dimension Type 4 - Historisierungstabelle

EmpID	Name	Standort	GueltigAb	GueltigBis
emp_1	Stefan Hilbrecht	Hamburg	01.01.1900	30.11.2021
emp_1	Stefan Hilbrecht	Berlin	01.12.2021	31.01.2022
emp_1	Stefan Hilbrecht	München	01.02.2022	31.12.2999

*Slowly Changing Dimension Type 6:* Diese Art der Historisierung ist eine Kombination aus den Arten SCD1, SCD2 und SCD3, woraus sich der Name dieser SCD ergibt. Wird ein neuer Datensatz hinzugefügt, erfolgt das Anlegen eines neuen Datensatzes wie in SCD2. Die Spalte des aktuellen Wertes eines Attributes wird, wie in SCD1, mit dem neusten Wert überschrieben. Zusätzlich wird eine Spalte mit den bisherigen Werten eines Attributes wie in SCD3 erstellt.

Tabelle 2.11: Slowly Changing Dimension Type 6 - Historisierung eines Datensatzes nach Änderung des Standort-Attributes

EmpID	Name	Aktueller Standort	Bisheriger Standort	GueligAb	GueligBis	Guelig
emp_1	Stefan Hilbrecht	Berlin	Hamburg	01.01.1900	31.01.2022	0
emp_1	Stefan Hilbrecht	Berlin	Berlin	01.02.2022	31.12.2999	1

## 2.4.2 Data Lake

Das Konzept des Data Lakes wurde erstmals im Jahre 2010 von James Dixon auf seinem Blog vorgestellt. Als einer der Gründer von Pentaho, einem Unternehmen, welches Business Intelligence-Software entwickelt, suchte Dixon eine Lösung für Unternehmen, die Daten mit den folgenden Eigenschaften behandelten:

- sind strukturiert oder semi-strukturiert
- stammen aus einer einzigen Quelle
- sind teil-transaktional oder nicht-transaktional
- benötigte Informationen aus diesen Daten sind teilweise bekannt
- weitere benötigte Informationen werden zu einem späteren Zeitpunkt bekannt
- es existieren unterschiedliche Benutzergruppen, die Zugriff auf die Daten benötigen
- es handelt sich um große Datenmengen

Dixon beschreibt einen Data Lake als einen See in einem natürlichen Zustand. Der See ist mit Daten aus einer Datenquelle gefüllt und unterschiedliche Nutzer können den See untersuchen, darin eintauchen oder eine Probe entnehmen. Data Marts (siehe Kapitel 2.4.3) werden als Wasserflaschen bezeichnet, die aufbereitetes, geeignetes Wasser in überschaubaren Mengen enthalten. [Dix10]

Im Laufe der Zeit haben sich unterschiedliche Definitionen und Varianten eines Data Lakes entwickelt. Nach [Ale] bezeichnet einen Data Lake als einen sehr großen Datenspeicher, welcher Daten unstrukturiert in ihrem Rohformat beinhaltet. Diese Daten stammen aus unterschiedlichen Quellen und durchlaufen keinerlei Transformationen, bevor sie im Data Lake abgelegt werden. Die Quellen, aus denen sie stammen, haben ständigen Zugriff auf den Data Lake und können die Dateien jederzeit ablegen. Dabei handelt es sich beispielsweise um Textdaten wie Emails, Kundenbewertungen oder Protokolle, Video- oder Bilddaten oder Sensordaten.

Der Vorteil dieser Art der Datenspeicherung ist, dass vor dem Ablegen der Daten der spätere Verwendungszweck nicht bekannt sein muss. Erst, wenn die Daten verwendet werden, werden die nötigen Transformationen durchgeführt. [Luba] Es wird also kein gewöhnlicher ETL-Prozess durchgeführt. Die Bestandteile des ETL-Prozesses werden in eine andere Reihenfolge gebracht: zu Beginn erfolgt die Extrahierung aus den Quellsystemen, dann das Laden in den Data Lake und zum Schluss die Transformationen. Daraus ergibt sich ein *ELT*-Prozess. [Talc]

Die Architektur eines Data Lakes gliedert sich nach [Are18] in drei Schichten:

### 1. Ingestion Layer

Diese Schicht dient als Speicherort für ankommende Daten. Um die Daten mit passenden Meta-Daten anzureichern, können die Daten geordnet abgelegt werden.

### 2. Caching Layer

In dieser Schicht werden bereits aufbereitete Daten (temporär) gespeichert. Diese Daten können für Auswertungen verwendet werden.

### 3. Processing Layer

Diese Schicht wird für die Analyse großer Datenmengen verwendet. Von dort kann auf die Daten in den vorherigen beiden Schichten zugegriffen werden.

Data Lakes müssen ein Data Warehouse nicht zwangsläufig ersetzen. Ein Data Lake kann anstelle eines Data Warehouses existieren, er kann jedoch auch zusätzlich zu einem solchen vorhanden sein. Dann dient er als ein Zwischenspeicher, in dem die Daten so lange gespeichert werden, bis klar ist, ob und in welcher Form sie weiterhin verwendet werden. [Sar]

Die Verwendung eines Data Lakes bringt jedoch auch Herausforderungen mit sich. Diese beginnen bereits mit dem Aufbau des Lakes. Es existieren unterschiedliche mögliche Architekturen, was die Entscheidung für die vorteilhafteste Variante für den jeweiligen Anwendungsfall erschwert. Zudem sind weder Bewertungen der Ansätze vorhanden noch Details zur Umsetzung der verschiedenen Architekturmodelle. Einige dieser Modelle widersprechen Dixons ursprünglichem Modell eines Data Lakes.

Um das bestmögliche Architekturmodell zu ermitteln, muss unternehmensintern eine Data Governance-Strategie festgelegt werden. Dabei müssen beispielsweise die unterschiedlichen Datenarten und deren Anforderungen an die Datenqualität überprüft werden.

Für eine optimale Umsetzung des Data Lakes ist eine ganzheitliche Strategie erforderlich, welche das Zusammenspiel der Data Lake-Komponenten untereinander sowie mit der Datenmodellierung berücksichtigt. [GGH<sup>+</sup>20]

Werden mehrere Datenquellen verwendet, muss darauf geachtet werden, dass die Daten strukturiert abgelegt und mit Metadaten angereichert werden, um die spätere Verwendung zu vereinfachen. Andernfalls würde die Verwendung der Daten einen hohen analytischen Aufwand voraussetzen, um ein Schema zu entwickeln, nach welchem die Daten aufbereitet werden können. [MR14]

## 2.4.3 Data Marts

Wird zur Datenhaltung nur eine zentrale Datenbank verwendet, von welcher aus alle Analysen durchgeführt werden, sorgt das für eine hohe Auslastung des Systems. Besonders, wenn sich die Anzahl der Nutzer oder die Menge der Daten erhöht, verschlechtert sich die Performanz. Die Lösung für dieses Problem ist die Verteilung der erforderlichen Aufgaben auf mehrere Datenbanken. Bei dieser Vorgehensweise ist die ausschließliche Verwendung der Ableitungsdatenbank für die Analysen nicht möglich. Stattdessen wurden Data Marts (deutsch: Auswertungsdatenbanken) entwickelt, die Ausschnitte des gesamten Datenbestandes darstellen. Die Architektur und Modellierung der Data Marts unterscheidet sich nicht von der einer Ableitungsdatenbank. [BG13, S. 67f] Die möglichen Gründe für die Verwendung von Data Marts sind vielseitig [KSS14, S. 36f]:

- Eigenständigkeit

Die Abteilungen eines Unternehmens sehen nur die für sie relevanten Daten.

- **Datenschutz**  
Die Daten sind nur für Nutzer sichtbar, denen der Umgang mit diesen gestattet ist.
- **Lastverteilung**  
Die Analysen werden auf die Data Marts aufgeteilt, um die Auslastung des Gesamtsystems zu reduzieren.
- **Datenvolumen**  
Es sind andere Realisierungsformen möglich, da in Data Marts weniger Daten enthalten sind als in der Ableitungsdatenbank.
- **Mobiler Zugang**  
Durch das geringere Volumen ist eine Nutzung auf mobilen Geräten möglich.

Das gewünschte Ziel bei der Verwendung von Data Marts ist also die vereinfachte Übersicht über die Daten sowie die Reduzierung der gesamten Daten auf die für den jeweiligen Anwendungsfall relevanten Daten. [BG13, S. 68]

Data Marts lassen sich in zwei Gruppen unterteilen: abhängige und unabhängige Data Marts. *Abhängige* Data Marts werden mit Daten aus der Ableitungsdatenbank befüllt. Bei der Übertragung werden keine Transformationen mehr durchgeführt, da die Daten im Data Mart zu denen in der Ableitungsdatenbank identisch sein sollen. Die Arten der Datenausschnitte lassen sich in strukturelle Extrakte (z.B. nur Kennzahlen oder Dimensionen einer bestimmten Berichtsgruppe), inhaltliche Extrakte (z.B. nur ein bestimmter Zeitraum oder eine bestimmte Region) und aggregierte Extrakte (z.B. geringere Granularität als in der Ableitungsdatenbank) kategorisieren. Auf abhängigen Data Marts ausgeführte Analysen bringen immer Ergebnisse, die konsistent zu auf der Ableitungsdatenbank durchgeführten Ergebnissen sind, da sie auf identischen Daten beruhen. [BG13, S. 68ff]

*Unabhängige* Data Marts entstehen durch einzelne Abteilungen, die ihre Daten in eigenen Datenbanken speichern und sie nicht in einem zentralisierten Datenspeicher ablegen. Der Vorteil dieser Vorgehensweise ist, dass die Abteilungen jederzeit Zugriff auf alle für sie nötigen Daten haben. Nachteilig ist jedoch, dass abteilungsübergreifende Analysen nur schwierig durchführbar sind, da die benötigten Daten vor der Analyse vereinheitlicht werden müssen. Dafür müssen Transformationen durchgeführt werden, weshalb die Daten in Data Marts und Ableitungsdatenbank nicht mehr identisch sind. Das führt außerdem dazu, dass die Analyseergebnisse, die auf den unterschiedlichen Datenbanken durchgeführt wurden, nicht miteinander übereinstimmen. [BG13, S. 70ff]

## 2.5 Analyse

Der Prozess der Analyse in einem BI-System dient dazu, aus den vorhandenen Daten Informationen zu erhalten, die als Basis für die Erkennung von Zusammenhängen im Unternehmen oder als Grundlage für Entscheidungen verwendet werden können. [BG13, S. 113] Nach [BG13, S. 113] lassen sich die Möglichkeiten der Datenanalyse in drei Kategorien einteilen:

- **Data Access**  
Mit Hilfe von SQL-Befehlen werden aggregierte Informationen aus der Datenbasis abgefragt, z.B. *Wie viele Mitarbeiter umfasste das Marketing-Team im Dezember 2021?*



- Online Analytical Processing

Mit Hilfe von OLAP werden komplexe Abfragen gestellt, deren Ergebnisse einen höheren Auswertungsbedarf haben, z.B. *Wie hat sich der Umsatz eines Teams im Vergleich zum Vormonat verändert?*.

- Data Mining

Die Daten werden auf Auffälligkeiten untersucht und in logische Zusammenhänge gebracht [BG13, S. 113] nach [DF95], z.B. *Welche Faktoren beeinflussen auf welche Weise die Entwicklung der Mitarbeiteranzahl im Marketing-Team?*.

Diese Analysemöglichkeiten können verwendet werden, um Berichte zu erstellen. Diese Berichte werden genutzt, um Risiken zu erkennen, zukünftige Entwicklungen abschätzen zu können oder einen Überblick über den derzeitigen Stand des Unternehmens zu erhalten. [KSS14, S. 273] nach [KBM10] Die Informationen können in textuellem, numerischem oder grafischem (Tabellen, Diagramme) Format dargestellt sein. [KSS14, S. 273]

Nach [KSS14, S. 273f] sind die erstellbaren Berichte in drei unterschiedliche Arten zu unterteilen:

- Standardberichte

In periodischen Zeitabständen (z.B. wöchentlich oder monatlich) wird automatisch ein Bericht zur regelmäßigen Information erstellt. Die Anfertigung kann auf Basis einer Vorlage erfolgen, sodass diese Berichte einem standardisierten Format folgen.

- Berichte zur Früherkennung

Bei Über- oder Unterschreitungen festgelegter Schwellwerte wird ein Bericht erstellt, der die Risiken dieser Grenzüberschreitungen verdeutlicht. Auf dieser Grundlage kann zeitnah reagiert werden.

- Ad-Hoc-Berichte

OLAP-Analysen werden visuell dargestellt, was zu individuellen Berichten führt, die dem aktuellen Informationsbedarf entsprechen.

## 3 Qualität in Softwareprodukten und BI-Systemen

### 3.1 Qualität von Softwareprodukten

#### 3.1.1 Qualitätskriterien

Für Qualität im Allgemeinen existiert keine eindeutige Definition, da vom Kontext abhängt, ob die Definition in diesem Zusammenhang eine passende Erklärung bietet. Nach dem Oxford English Dictionary bezeichnet Qualität den "Grad der Exzellenz"[AFF90], nach den Normen in ISO 9000:2008 ist Qualität die "Gesamtheit der Merkmale und Eigenschaften eines Produkts oder einer Dienstleistung, die sich auf seine Fähigkeit auswirken, bestimmte oder implizierte Bedürfnisse zu befriedigen". [ISO08b] Nach ISO 9001:2008 [ISO08c] wird Qualität ermittelt, indem die Merkmale einer Sache mit den Anforderungen an diese abgeglichen werden. Sind die Anforderungen erfüllt, deutet das auf eine sehr gute Qualität hin; sind sie es nicht, ist die Qualität niedrig. [Gil11, S. 5]

Die Schwierigkeit, eine Definition für Qualität zu entwickeln, liegt darin begründet, dass Qualität nicht mit physischen Attributen gemessen werden kann, wie etwa die Temperatur oder die Länge eines Gegenstandes. Qualität wird durch mehrere Eigenschaften bestimmt, von denen einige mit einfachen Methoden gemessen werden können, andere hingegen nur sehr aufwändig zu bestimmen sind. Um welche konkreten Eigenschaften es sich handelt, ist abhängig von der Art der zu beurteilenden Sache. Der Grad der Qualität ist aus Sicht der Kunden oft vom Preis der Ware abhängig. Ein überdurchschnittlich hoher Preis erzeugt sehr hohe Erwartungen an die Qualität; für durchschnittliche Qualität sind die Kunden nur einen durchschnittlichen Preis zu zahlen bereit. Soll also ein preiswertes Produkt entwickelt werden, müssen die Eigenschaften priorisiert werden, um die Herstellungskosten gering zu halten. Einige Qualitätskriterien sind dabei leichter verzichtbar als andere und einige beeinflussen sich gegenseitig. [Gil11, S. 4]

Die Übertragung des Qualitätsbegriffes auf Softwareprodukte gestaltet sich schwierig, da Software keine physische Erscheinungsform hat und sich zudem Teile der Software oder der verwendeten Hardware schnell ändern können.

Zudem sind häufig die oft sehr hohen Anforderungen der Kunden zu Beginn des Projektes noch nicht vollständig bekannt und können sich im Laufe des Entwicklungsprozesses ändern.

Der Wirtschaftswissenschaftler David Alan Garvin entwickelte 1984 ein Qualitätsmodell, in welchem fünf Sichtweisen auf Qualität dargestellt sind. [Gil11, S. 11-15] nach [Gar84] Diese lassen sich auch auf die Softwarequalität übertragen:

1. Transzendente Sicht  
Diese Sichtweise lässt sich nicht messen und nur schwierig auf ein Softwareprojekt übertragen. Sie beschreibt die Wirkung, die ein Produkt auf einen Nutzer hat.
2. Produktbezogene Sicht  
Die Verbesserung der Qualität hat einen höheren Preis zur Folge. Die Steigerung der Qualität kann durch bessere Funktionalität (z.B. mehr Sicherheit durch bestimmte Funktionen) oder durch die gewissenhaftere Entwicklung der Software (z.B. durch Verwendung eines Qualitätsmanagementsystems) erfolgen.
3. Nutzerbasierte Sicht  
Die Softwarelösung muss passend zu dem Problem des Nutzers sein, das mit der Software gelöst werden soll.
4. Produktionssicht  
Diese Sichtweise betrachtet, inwieweit die Eigenschaften der Software mit den Anforderungen übereinstimmen.
5. Wertorientierte Sichtweise  
Die Software soll die Anforderungen der Nutzer erfüllen und gleichzeitig deren Budget nicht übersteigen.

Nach den Normen ISO/IEC 9126 über die Qualität von Softwareprodukten ist Softwarequalität definiert als die „Gesamtheit der Merkmale und Merkmalswerte eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen.“ [ISO01] Auch nach dieser Definition sind mehrere Qualitätskriterien nötig, um die Softwarequalität zu bestimmen. Es existieren unterschiedliche Theorien, welche Kriterien zur Beurteilung der Qualität nötig sind. Tabelle 3.1 enthält eine Übersicht über die Kriterien, die in bekannten Qualitätsmodellen enthalten sind.

Tabelle 3.1: Qualitätskriterien nach [Gil11, S. 24,32,35], [Onp] nach [ISO01], [ISO11], [MRW77], [BBL76]

Kriterium	Erklärung
Funktionalität	Ausmaß, in dem die Software die an sie gestellten Anforderungen erfüllt
Zuverlässigkeit	Fehlerfreier Nutzung der Software
Änderbarkeit / Verständlichkeit / Transparenz	Einfachheit, den Code der Software zu verstehen, um Wartungen und Anpassungen durchzuführen
Übertragbarkeit	Aufwand, der nötig ist, um die Software in einer anderen Umgebung zu verwenden
Benutzbarkeit	Einfachheit der Benutzung
Effizienz / Performanz	Effiziente Ausführung und Speichernutzung
Wartbarkeit	Aufwand, der nötig ist, um einen Fehler zu erkennen und zu beheben

Tabelle 3.1: Qualitätskriterien nach [Gil11, S. 24,32,35], [Onp] nach [ISO01], [ISO11], [MRW77], [BBL76]

Kriterium	Erklärung
Kompatibilität	Aufwand, der nötig ist, um die Software mit einem anderen System zu verbinden
Testbarkeit	Einfachheit, mit der Softwaretests durchgeführt werden können
Humantechnik	Einbezug menschlicher Fähigkeiten und Bedürfnisse in die Planung von Prozessen und Produkten
Sicherheit / Integrität	Schutz vor unberechtigtem Zugriff
Wiederverwendbarkeit	Einfachheit, mit der die Software in einem anderen Kontext wiederverwendet werden kann
Korrektheit	Maß, in welchem die Anforderungen erfüllt werden
Flexibilität	Einfachheit, mit der Änderungen an der Software vorgenommen werden können

Die Bestimmung der Erfüllung der Qualitätskriterien erfolgt mit Hilfe von Metriken. Eine Metrik ist „eine messbare Eigenschaft, die ein Indikator für eines oder mehrere der Qualitätskriterien ist, die wir zu messen versuchen.“ [Gil11, S. 41] Eine Metrik muss einen klaren Bezug zu den zu messenden Kriterien haben, die verschiedenen Ausmaße der Kriterien berücksichtigen sowie die objektive Beurteilung dieser auf einer Skala abbilden können. Die Unterscheidung erfolgt dabei in vorhersagende Metriken, die verwendet werden, um Prognosen über die Softwarequalität im weiteren Lebenszyklus der Software zu stellen oder beschreibende Metriken, welche Bewertungen der aktuellen Softwarequalität vornehmen. [Gil11, S. 41ff]

Nach ISO 9126 lassen sich Metriken weiter in externe, interne und In-Use-Metriken einteilen. Die internen Metriken werden während der Entwicklung eines Software-Projektes eingesetzt, um die Qualität des aktuellen Entwicklungsstandes zu prüfen sowie Vorhersagen über die Qualität des fertigen Produktes zu treffen, um rechtzeitig qualitätsverbessernde Entscheidungen treffen zu können. Externe Metriken messen die Qualität der Software während dem Testen sowie der Ausführung in der Zielumgebung. In-Use-Metriken messen die Qualität ebenfalls während der Ausführung. Am Beispiel des Kriteriums der Funktionalität trifft die interne Metrik Prognosen, wie viele der vorhandenen Funktionen nicht mit den geplanten Anforderungen übereinstimmen. Die konkrete Anzahl dieser Funktionen wird durch externe Metriken bestimmt. Die In Use Metrik orientiert sich an den Erwartungen der Nutzer - sie bestimmt die Anzahl der Funktionen, welche deren Erwartungen nicht erfüllen. [Gil11, S. 53]

Im Jahre 1977 entwickelte J. McCall ein Modell für die Bestimmung von Softwarequalität. Dieses enthält 11 Kriterien, die in drei Kategorien unterteilt wurden. Diese Kategorien sind der Betrieb, die Überarbeitung sowie die Umstellung eines Produktes. [Gil11, S. 22f] Abbildung 3.1 enthält alle von McCall entwickelten Kriterien.

Während des *Betriebes eines Produktes (Product Operation)* ist es wichtig, dass das Produkt einfach erlernbar ist, effizient arbeitet und den Erwartungen des Nutzers entspricht. Bei der *Überarbeitung des Produktes (Product Revision)* werden darin enthaltene Fehler korrigiert sowie

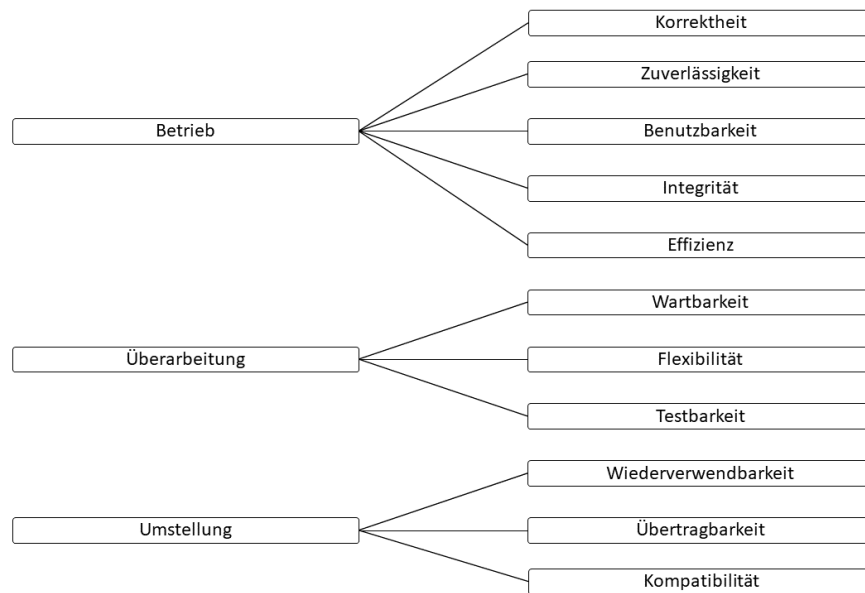


Abbildung 3.1: Qualitätskriterien nach McCall nach [Alk14]

gewünschte Anpassungen vorgenommen. Die Kriterien der *Produktumstellung (Product Transition)* liegen in einer immer größer werdenden Wichtigkeit der verteilten Verarbeitung sowie der regelmäßigen Erneuerung von Hardware begründet.

Ein weiteres Qualitätsmodell wurde 1978 von B. Boehm entwickelt (Abbildung 3.2). Dieses basiert auf McCalls Modell, unterscheidet sich jedoch darin, dass es um einige Kriterien erweitert wurde. McCalls Modell fokussiert sich auf sogenannte *high-level Attribute*, welche im Boehms Modell unter den *As-is*-Eigenschaften aufgeführt sind. Boehm erweiterte diese um Eigenschaften der Kategorien *Wartbarkeit (Maintainability)* und *Portability (Übertragbarkeit)*. [Gil11, S. 25f] [Har]

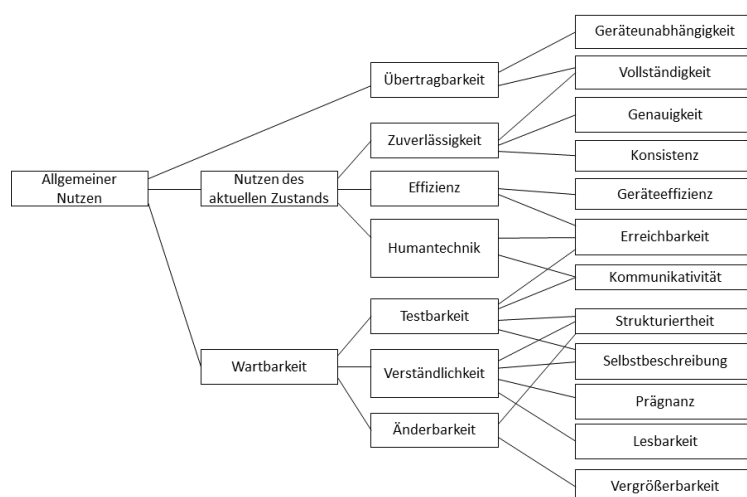


Abbildung 3.2: Qualitätskriterien nach Boehm ([Gil11, S. 26] nach [BBL76])

Im Jahre 1991 wurde erstmals eine ISO/IEC-Norm für die Kriterien der Softwarequalität entwickelt. Bei dieser Norm handelt es sich um ISO/IEC 9126, welche unter dem Namen „Software engineering - Product quality“ veröffentlicht wurde. Die enthaltenen Eigenschaften beruhen auf den Modellen von McCall und Boehm, wurden jedoch in sechs Kategorien unterteilt. (Abbildung 3.3 [Har]

<b>Funktionalität</b> <ul style="list-style-type: none"> <li>• Eignung</li> <li>• Genauigkeit</li> <li>• Kompatibilität</li> <li>• Sicherheit</li> <li>• Einhaltung der Funktionalität</li> </ul>	<b>Zuverlässigkeit</b> <ul style="list-style-type: none"> <li>• Laufzeit</li> <li>• Fehlertoleranz</li> <li>• Wiederherstellbarkeit</li> <li>• Einhaltung der Zuverlässigkeit</li> </ul>	<b>Benutzbarkeit</b> <ul style="list-style-type: none"> <li>• Verständlichkeit</li> <li>• Lernfähigkeit</li> <li>• Betriebsfähigkeit</li> <li>• Attraktivität</li> <li>• Einhaltung der Benutzbarkeit</li> </ul>
<b>Effizienz</b> <ul style="list-style-type: none"> <li>• Zeitverhalten</li> <li>• Ressourcenauslastung</li> <li>• Einhaltung der Effizienz</li> </ul>	<b>Wartbarkeit</b> <ul style="list-style-type: none"> <li>• Analysierbarkeit</li> <li>• Veränderbarkeit</li> <li>• Stabilität</li> <li>• Testbarkeit</li> <li>• Einhaltung der Wartbarkeit</li> </ul>	<b>Übertragbarkeit</b> <ul style="list-style-type: none"> <li>• Anpassungsfähigkeit</li> <li>• Installierbarkeit</li> <li>• Koexistenz</li> <li>• Austauschbarkeit</li> <li>• Einhaltung der Übertragbarkeit</li> </ul>

Abbildung 3.3: Qualitätskriterien ISO/IEC 9126 [AQ] nach [ISO01]

20 Jahre nach der Veröffentlichung der ISO/IEC 9126 wurde sie durch ISO/IEC 25010 ersetzt, welche den Titel „System- und Softwaretechnik - System- und Softwarequalitätsanforderungen und -bewertung (SQuaRE) - System- und Softwarequalitätsmodelle“ trägt. Aktuell erfolgt eine Aufteilung der ISO/IEC 25010:2011 in ISO/IEC CD 25002.2 (Qualitätsmodelle - Überblick und Verwendung), ISO/IEC CD 25010 (Modell der Produktqualität) sowie in ISO/IEC CD 25019.2 (Qualität im Nutzungsmodell). [ISO11] Die folgenden (Abbildung 3.4) enthaltenen Kriterien sind als Nachfolger der Kriterien aus ISO/IEC 9126 zu betrachten.

<b>Funktionalität</b> <ul style="list-style-type: none"> <li>• Funktionale Vollständigkeit</li> <li>• Funktionale Korrektheit</li> <li>• Funktionale Angemessenheit</li> </ul>	<b>Kompatibilität</b> <ul style="list-style-type: none"> <li>• Koexistenz</li> <li>• Interoperabilität</li> </ul>	<b>Zuverlässigkeit</b> <ul style="list-style-type: none"> <li>• Reife</li> <li>• Verfügbarkeit</li> <li>• Fehlertoleranz</li> <li>• Wiederherstellbarkeit</li> </ul>	<b>Wartbarkeit</b> <ul style="list-style-type: none"> <li>• Modularität</li> <li>• Wiederverwendbarkeit</li> <li>• Analysierbarkeit</li> <li>• Modifizierbarkeit</li> <li>• Testbarkeit</li> </ul>
<b>Performanz</b> <ul style="list-style-type: none"> <li>• Zeitverhalten</li> <li>• Ressourcenverwendung</li> <li>• Kapazität</li> </ul>	<b>Gebrauchstauglichkeit</b> <ul style="list-style-type: none"> <li>• Angemessenheitserkennung</li> <li>• Erlernbarkeit</li> <li>• Operabilität</li> <li>• Fehlersicherheit</li> <li>• Ästhetik der Benutzerschnittstelle</li> <li>• Barrierefreiheit</li> </ul>	<b>Sicherheit</b> <ul style="list-style-type: none"> <li>• Vertraulichkeit</li> <li>• Integrität</li> <li>• Nachweisbarkeit</li> <li>• Verantwortlichkeit</li> <li>• Authentizität</li> </ul>	<b>Übertragbarkeit</b> <ul style="list-style-type: none"> <li>• Anpassbarkeit</li> <li>• Installierbarkeit</li> <li>• Austauschbarkeit</li> </ul>

Abbildung 3.4: Qualitätskriterien ISO/IEC 25010 nach [ISO11]

Nach Dirk W. Hoffmann [Hof13] ist eine Einteilung der Qualitätskriterien nach dem Nutzen für den Kunden oder den Hersteller sinnvoll. Kundenorientierte Kriterien sind diejenigen, welche die Qualitätssicht des Anwenders symbolisieren und die Kaufentscheidungen des Kunden beeinflussen. Dazu zählen die Funktionalität, Performanz, Zuverlässigkeit und Benutzbarkeit. Herstellerorientierte Kriterien sind nicht auf den ersten Blick erkennbar, jedoch für den langfristigen Erfolg des Produktes verantwortlich. Dazu gehören die Wartbarkeit, Transparenz, Übertragbarkeit und Testbarkeit eines Softwareproduktes.

Alle Kriterien in einem hohen Maß zu erfüllen, ist nicht möglich, da sich einige gegenseitig ausschließen. Ein Beispiel dafür sind die Performanz und die Übertragbarkeit. Um übertragbar zu sein, können von der Software nur Funktionen verwendet werden, die auf allen Plattformen zur Verfügung stehen. Um performant zu sein, muss die Software die Vorteile der ausgewählten Architektur nutzen, die jedoch möglicherweise auf anderen Plattformen nicht verfügbar sind. Somit beeinflussen sich Performanz und Übertragbarkeit gegenseitig negativ. [Hof13, S. 10ff] Abbildung 3.5 enthält eine Übersicht über die Verhältnisse verschiedener Qualitätskriterien untereinander.

	Laufzeit	Zuverlässigkeit	Benutzbarkeit	Transparenz	Übertragbarkeit	Wartbarkeit	Testbarkeit
Funktionale Korrektheit	-	+		+	+	+	+
Laufzeit		-		-	-	-	-
Zuverlässigkeit				+			+
Benutzbarkeit							
Transparenz					+	+	+
Übertragbarkeit						+	
Wartbarkeit							+

Abbildung 3.5: Korrelationsmatrix der Qualitätskriterien nach [Hof13, S. 11]

Auffällig dabei ist, dass die Laufzeit dabei zu allen Kriterien (bis auf die Benutzbarkeit) in negativer Verbindung steht. Die Benutzbarkeit dagegen beeinflusst keines der anderen Kriterien - die Benutzbarkeit lässt sich also perfektionieren, ohne negative Konsequenzen auf die Software auszuüben.

Auch die Messung dieser Kriterien mit Hilfe von Metriken gestaltet sich aus verschiedenen Gründen schwierig, beginnend bei der Festlegung des Verhältnisses des Wertes auf der Metrik zu dem tatsächlichen Wert der Softwarequalität. Die Tatsache, dass es keine festen Grenzen gibt, die die schlechtest- und bestmögliche Qualität festlegen sowie die Subjektivität von Kriterien wie der Benutzbarkeit erschweren eine Bewertung zusätzlich. Weiterhin existiert nicht für jedes Kriterium eine entsprechende Metrik; andere wiederum messen mehr als nur ein Kriterium. [Gil11, S. 52f]

Die Anwendung dieser Kriterien ist jedoch nicht nur auf OLTP-basierender Software möglich, sondern auch auf BI-Systeme. Die Kriterien sind dabei nichtzwangsläufig als gleichermaßen wichtig zu betrachten. BI-Systeme werden in der Regel nicht als Standardprodukte verkauft, sondern

für jeden Anwendungsfall individuell entwickelt, weshalb die Architektur, auf welcher es angewendet werden soll, von Beginn an bekannt ist, sodass die Komponenten von Beginn an optimal und passgenau entwickelt werden können. Deshalb sind die *Kompatibilität* und die *Übertragbarkeit* bei der Qualitätsprüfung von BI-Systemen zu vernachlässigen. Die *Sicherheit* eines BI-Systems wird zu einem großen Teil durch den Umgang mit personenbezogenen Daten bestimmt. In diesem Rahmen ist eine Festlegung notwendig, welche Daten zwingend gespeichert werden müssen sowie welche Mitarbeiter mit welchen Daten arbeiten dürfen. Dabei ist sicherzustellen, dass ausschließlich berechnete Mitarbeiter die Möglichkeit des Zugriffs auf diese Daten haben. Die *Gebrauchstauglichkeit* des Systems ist individuell an jede Umgebung anzupassen. Zu beachten ist dabei beispielsweise, wie geübt die Nutzer im Umgang mit Technik sind und auf welchen eventuellen Einschränkungen der Nutzer dabei besonders eingegangen werden muss. Durch dieses hohe Maß an Individualität wird die Gebrauchstauglichkeit in diesem Zusammenhang nicht weiter betrachtet. Der *Performanz* eines BI-Systems lässt sich im Vergleich zu den anderen Kriterien eine geringe Wichtigkeit zuordnen. Zentral ist dabei die Frage, ob der Nutzen einer korrekten Berechnung höher zu priorisieren ist als die Dauer des Berechnungsprozesses. Da eingangs erläutert wurde, welches Ausmaß fehlerhafte Berechnungen in BI-Systemen haben können sowie die eher regelmäßige als spontane Nutzung der Berichterstellung, kommt hier der Performanz eine eher geringe Wichtigkeit zu.

Im Vordergrund stehen die verbleibenden Kriterien aus ISO/IEC 25010, *Funktionalität*, *Zuverlässigkeit* und *Wartbarkeit*. Wartbarkeit ist ein wichtiges Kriterium, um zu gewährleisten, dass das System auch bei wachsendem technologischen Fortschritt auf dem neusten Stand gehalten werden kann. Dafür ist eine gute vorherige Planung nötig, welche beispielsweise eine sinnvolle Modularisierung enthält. Wartbarkeit wird zudem teilweise von funktionalen Tests abgedeckt, da ein Bestandteil dieser die Testbarkeit ist. Die Zuverlässigkeit ist ebenso ein wichtiges Kriterium, da beispielsweise die Verfügbarkeit ausschlaggebend für den Erfolg der Anwendung ist. Die Funktionalität ist der Kern des BI-Systems, da diese für das Beinhalten aller nötigen Funktionen sowie deren korrekte Ausführung steht. Da ohne eine sichergestellte, korrekte Funktionalität die anderen Kriterien keine Anwendung finden können, liegt der Fokus dieser Arbeit auf der Sicherstellung der Funktionalität von BI-Systemen.



### 3.1.2 Sicherstellung der Qualität

Die Möglichkeiten, mit denen sich die Qualität einer Software messen und verbessern lässt, sind zahlreich. Die Abbildung 3.6 stellt die nach [Hof13] existierenden Maßnahmen dar.

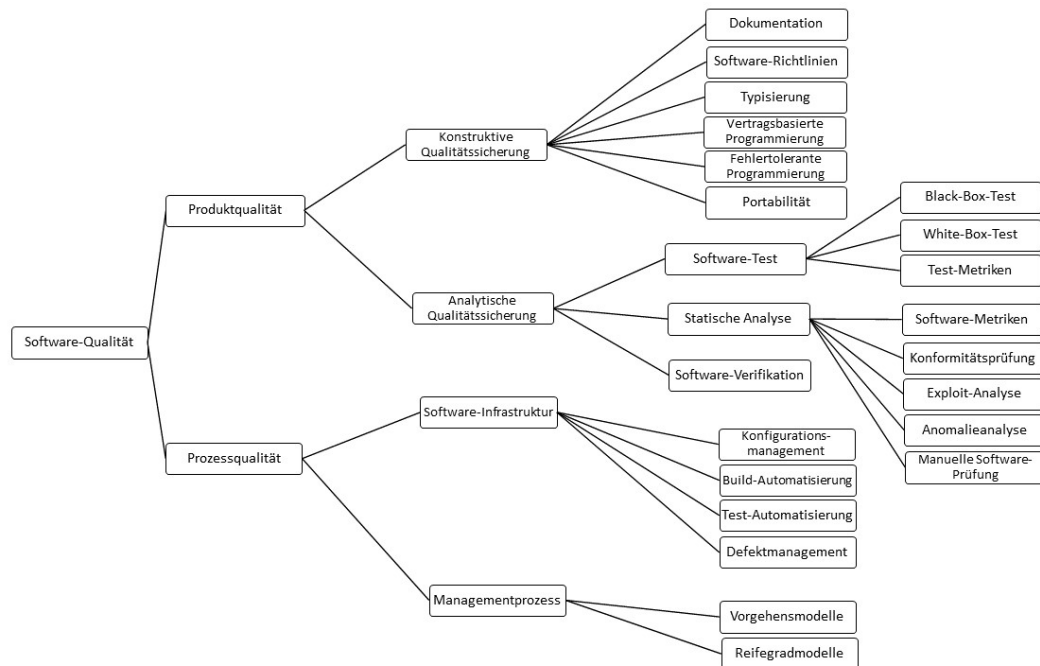


Abbildung 3.6: Qualitätssicherungsmaßnahmen nach [Hof13, S. 20]

Die erste Unterteilung erfolgt in die Sicherung der Produktqualität und die der Prozessqualität. Die Sicherstellung der Prozessqualität fokussiert sich auf die Entstehung der Software, also alle Vorgaben, die getroffen werden, um einen Prozess festzulegen, welcher hohe Qualitätsstandards ermöglichen kann. Die Unterteilung der Prozessqualität erfolgt in die Bereiche der Software-Infrastruktur, welche aus Entwicklersicht fundamentale Bedeutung hat und zu welcher unter Anderem die Test-Automatisierung gehört, sowie in die Managementprozesse, die besonders für die Führungsebene bedeutend sind. [Hof13, S. 25] Die Produktqualität konzentriert sich auf das Softwareprodukt, das entstehen soll und damit verbunden auf die Erfüllung der in Kapitel 3.1.1 erläuterten Qualitätskriterien. Unterteilt wird hier in die konstruktive sowie in die analytische Qualitätssicherung. [Hof13, S. 19f] Die konstruktive Qualitätssicherung meint die Vorbereitung und begleitende Aktivitäten ergänzend zur Entwicklung des geplanten Produktes durch beispielsweise Richtlinien, Templates sowie Vorgehensmodelle. Die analytische Qualitätssicherung dient der Sicherstellung der Qualität eines Produktes, nachdem dessen Entwicklung abgeschlossen ist. [pmq] Da sich diese Form der Qualitätssicherung direkt mit dem entstandenen Produkt befasst, ist sie am geeignetsten, um die Übertragung auf Business Intelligence-Systeme zu untersuchen. Im weiteren Verlauf der Arbeit liegt der Fokus auf Softwaretests, da diese die am weitesten verbreitete Art der analytischen Qualitätssicherung darstellen.

Das Prinzip der Tests besteht darin, die zu überprüfende Software mit bestimmten Eingabedaten auszuführen und das tatsächliche Verhalten der Software mit dem erwarteten Verhalten abzugleichen. [Hof13, S. 157] Dabei sollen Verhaltensweisen aufgedeckt werden, die auf Fehlerzustände hindeuten. Nachdem auf diese Art ermittelte Fehler behoben wurden, können die selben Tests verwendet werden, um sicherzustellen, dass die Software sich nun korrekt verhält. [Sch20, S. 16ff]

Bei der Durchführung von Tests kann die Software daraufhin überprüft werden, ob sie sich bei Eingabe erlaubter Werte korrekt verhält (Positivtest) und ob sie im Falle der Eingabe unerlaubter Werte die vorgesehenen Fehlermeldungen ausgibt (Negativtest). Um alle möglichen Eingabewerte abzudecken, können Äquivalenzklassen gebildet werden (Äquivalenzklassenanalyse), welche die Anzahl der benötigten Testfälle stark reduzieren können. Ebenso ist eine Testung unter Verwendung der Werte, die die Grenzen der Äquivalenzklassen bilden (Grenzwertanalyse), zu empfehlen. [Sch20, S. 408]

Wird ein Test durchgeführt und zeigt das gewünschte Ergebnis, wird der Test mit *pass* bewertet; andererseits mit *fail*. Tritt ein Fehler auf, kann die Ursache dafür Fehlverhalten, ein Spezifikationsfehler, eine Spezifikationslücke oder ein Fehler im Testsystem sein.

Wie aufwändig die Tests gestaltet werden, wird davon abhängig gemacht, wie groß der Schaden sein kann, der im Falle eines Softwarefehlers anfallen kann. Die Anzahl der möglichen Testfälle ist für jede Softwarekomponente sehr hoch, sodass für gewöhnlich nicht alle Fehler ermittelt werden können, die während der Nutzung des Softwareproduktes auftreten können. Daher gilt, dass mit steigender Komplexität des Programmes auch der Testaufwand steigt. Wie viele der Anforderungen mit Hilfe von Tests überprüft werden, wird durch die sogenannte Testabdeckung angegeben. [Sch20, S. 16ff] Um einen Softwaretest zu erstellen, ist es nicht zwingend notwendig, den gesamten Code der zu testenden Funktion zu kennen. Die Dokumentation enthält im Normalfall die erlaubten Eingabewerte sowie das erwartete Verhalten nach anschließender Ausführung. Auf dieser Basis werden die sogenannten *Black-Box-Tests* entwickelt. [Sch20, S. 419] Ist hingegen der Code bekannt, werden *White-Box-Verfahren* angewendet. Bei diesem Testverfahren stehen die inneren Strukturen und die Art der Umsetzung im Fokus. [Sch20, S. 433]

Um die Software auf alle möglichen Arten testen zu können und damit eine höchstmögliche Testabdeckung zu erreichen, existieren verschiedene Arten der Softwaretests: [Sch20, S. 30-35]

- **Strukturtest**

Strukturtests lassen sich in datenflussorientierte sowie kontrollflussorientierte Tests einteilen. Datenflussorientierte Tests beziehen sich auf den Zugriff auf und die Veränderung von Variablen im Laufe des Programms. Kontrollflussorientierte Tests hingegen fokussieren sich auf Strukturelemente im Code, wie beispielsweise Verzweigungen oder Bedingungen. Um eine möglichst hohe Testabdeckung zu erreichen, kann das Ziel sein, jedes der Strukturelemente zu durchlaufen. Dadurch kann jedoch nicht die gesamte Funktionalität geprüft werden, sondern nur die korrekte Verwendung der Strukturelemente. Eine automatisierte Testdurchführung ist bei dieser Testmethode einfach zu erreichen, da die benötigten Daten für Programmabläufe, die alle Strukturelemente durchlaufen, sowie der Vergleich mit dem erwünschten Ergebnis, durch das Testsystem durchgeführt werden können.

- **Statistischer Test**

Die Testdaten, die für diese Testmethode verwendet werden, werden entweder zufällig generiert oder als realistische Anwendungsfälle entwickelt. Dadurch ist eine rechtzeitige Erkennung von Fehlern, die bei der Verwendung durch den Kunden auftreten würden, möglich. Der Nachteil dieser Vorgehensweise besteht in der schlechteren Erkennbarkeit von Fehlern in weniger häufig auftretenden Anwendungsfällen. Die Automatisierung statistischer Tests ist nicht möglich, da eine Erkennung des gewünschten Ergebnisses durch das Testsystem nicht leistbar ist.

- **Mutationentest**

Mit Mutationentests werden keine direkten Bestandteile der Software überprüft, sondern

die Qualität der weiteren durchgeführten Tests. Dafür wird der Code der Software stellenweise leicht modifiziert, um Fehler einzubauen. Die vorhandenen Tests werden ausgeführt und die fehlerhaften Teile der Software sollen durch die ausgeführten Tests erkannt werden. Diese Methode kann einfach automatisiert werden.

- Evolutionärer Test

Evolutionäre Tests beschreiben die automatische Entwicklung von Testfällen, welche eine festgelegte Zeit lang modifiziert werden, bis durch diese ein Softwarefehler ermittelt werden kann. Tritt in dieser Zeit kein Fehler auf, wird der erwartete Fehler als nicht vorhanden eingestuft. Da die Modifizierung der Testfälle automatisch erfolgt, ist kein großes Wissen über den Code der Software nötig. Selbst das Testen komplexer Funktionen ist so mit wenig Aufwand machbar.

- Funktionaler Test

Die funktionalen Tests stellen die wichtigste Testgruppe der dynamischen Tests dar. Mit diesen wird „das von außen beobachtbare Verhalten des Testobjekts geprüft“. Bei dieser Testmethode werden die Spezifikationen der zu testenden Software ermittelt, um anschließend für eine Anforderung einen Test erstellen zu können. Die Schwierigkeit dabei besteht darin, den Testumfang festzulegen. Dazu gehört, ob jede Anforderung getestet werden muss, ob ein einzelner Test ausreichend ist oder ob verschiedene Szenarien getestet werden sollten oder ob die Anforderungen einzeln oder kombiniert getestet werden müssen. Deshalb lässt sich nur schwierig beurteilen, wie viele und welche Tests für die Software ausreichend sind. Um die Anzahl der Tests gering zu halten und gleichzeitig möglichst viele Anwendungsfälle abzudecken, ist die Klärung dieser Fragen vor der Implementierung der Tests sinnvoll. Wurde ein Test durchgeführt, ohne dass ein Fehler erkannt wurde, wird der getestete Teil der Software als fehlerfrei eingestuft. Die Funktion erfüllt somit die entsprechende Spezifikation. Dass die Software damit den Anforderungen der Kunden genügt, kann jedoch nur sichergestellt werden, wenn alle Spezifikationen korrekt aufgestellt wurden. Diese Überprüfung muss jedoch durch statische Analysen wie beispielsweise Reviews erfolgen.

Software ist in Teilsysteme untergliedert, welche ebenfalls aus mehreren Komponenten bestehen. Dieser wiederum bestehen ebenfalls aus mehreren Funktionen. Jede dieser Architekturstufen kann getestet werden, weshalb eine Untergliederung der Testmöglichkeiten in folgende Teststufen erfolgen kann [Sch20, S. 54-64]

- Unit-Test

Diese Testmethode ist der erste Schritt bei der Erstellung von Softwaretests. Dabei wird die Implementierung einzelner Funktionen/Methoden getestet und überprüft, ob die Ein- und Ausgaben korrekt verarbeitet werden.

- Komponententest

Aus den einzelnen Funktionen werden sogenannte Komponenten entwickelt. Das Ziel der Komponententests ist es, sicherzustellen, dass die Funktionen innerhalb der Komponenten untereinander fehlerfrei arbeiten. Die Abläufe der Funktionen im Test sind dabei an Abläufen orientiert, die bei der geplanten Verwendung der Software regelmäßig auftreten werden. Typische Fehler, die bei Komponententests entdeckt werden, sind Berechnungsfehler oder fehlerhafte bzw sogar fehlende Programmpfade. Die Qualitätskriterien, die durch diese

Testmethode beeinflusst werden, sind Funktionalität, Robustheit, Effizienz, Performance und Wartbarkeit.

- Integrationstest

Bei dieser Testart wird das Zusammenspiel verschiedener Komponenten untereinander getestet. Ziel ist das Ermitteln von Fehlern in Schnittstellen der Komponenten untereinander, um eine funktionierende Kommunikation derer zu sichern. Auch hier erfolgt vorrangig das Testen gewöhnlicher Funktionsabläufe.

Erfolgt das Testen von Schnittstellen zwischen Komponenten und externen Softwaresystemen oder zu Hardware, handelt es sich um einen Systemintegrationstest.

- Systemtest

Bei einem Systemtest wird die Leistung des gesamten Systems überprüft, das heißt, wie gut das System die funktionalen und nicht-funktionalen Anforderungen aus Sicht des späteren Anwenders erfüllt. Die Tests erfolgen jedoch noch nicht auf der Zielumgebung, sondern in einer realistischen Testumgebung. Bei diesem Test sollen Fehler durch falsch, unvollständig oder widersprüchlich umgesetzte Anforderungen aufgedeckt werden.

- Produkttest

Diese Testmethode ist ähnlich dem Systemtest, mit dem Unterschied, dass die Tests nicht in einer Testumgebung, sondern in der Zielumgebung durchgeführt werden.

- Abnahmetest

Diese Tests beinhalten häufige Anwendungsfälle und sollen dem Kunden als Basis dienen, zu entscheiden, ob er das fertige Produkt akzeptiert oder ablehnt. Überprüft werden die Funktionalität sowie interne Qualitätsmerkmale wie die Einhaltung von Coderichtlinien und Standards. Die Basis für die Erstellung der Testfälle können beispielsweise vereinbarte Anwender- und Systemanforderungen, Lastenhefte, Geschäftsprozesse oder Risikoanalysen sein. Im Optimalfall wurden die Akzeptanztestfälle durch den Kunden selbst erstellt bzw. überprüft. Schwerwiegende Fehler, die erst in dieser Teststufe auffallen, sind häufig nicht mehr änderbar, da der Aufwand der Änderung zu groß wäre.

- Produktionstest

An dieser Stelle sind die funktionalen Tests bereits abgeschlossen. Der Produktionstest dient der Sicherung der Produzierbarkeit und dem Ausschluss von Produktionsfehlern im vollständig entwickelten Produkt.

- Feldtest

Die Software wird in unterschiedlichen Umgebungen getestet, indem Mitarbeiter des Herstellers (Alpha-Test) sowie ausgewählte Kunden (Beta-Test) die Software vorab verwenden dürfen und dem Hersteller mitteilen, wenn während der Nutzung Fehler aufgetreten sind.

- Regressionstest

Während der Weiterentwicklung, der Wartung oder im normalen Lebenszyklus der Software werden bereits vorhandene Testfälle nach einer Modifikation durchgeführt, um sicherzustellen, dass die Software nun fehlerfrei ist (*Verifikation der Fehlerbehebung*).

## 3.2 Qualität in BI-Systemen

Nach [ASD14] lassen sich die Kriterien aus ISO 25010 [ISO11] auf BI-Systeme genauso anwenden wie auf klassische Softwareprodukte.

Das Kriterium der Datenqualität ist gesondert zu betrachten. Die zu testenden Komponenten sind in diesem Fall diejenigen, die für den Datenspeicher zuständig sind. Eine Unterteilung der Tests nach den einzelnen Teststufen ist nicht erforderlich. Durch die Komplexität, die aus der Abdeckung aller Komponenten mit Testfällen entsteht, ist es notwendig, zu priorisieren, welche Subsysteme und Qualitätsmerkmale vorrangig getestet werden sollen.

Die verschiedenen Bestandteile eines BI-Systems lassen sich als Subsysteme betrachten, welche sich auf Basis des in Kapitel 2 erläuterten Aufbaus in den ETL-Prozess, das Data Warehouse, den Data Lake, Data Marts und Analyseanwendungen einteilen lassen. Diese Subsysteme können einzeln als Testobjekte betrachtet werden. [TZ16, S. 119-121] Da für den Betrieb eines BI-Systems zusätzlich zu den Subsystemen die Anbindung an die Quellsysteme von elementarer Bedeutung ist, sind diese Schnittstellen ebenfalls eine zu testende Komponente des Systems. [TZ16, S. 122] nach [SBGB12, S. 92] Im Rahmen dieser Arbeit werden diese Schnittstellen jedoch nicht betrachtet, da die Verantwortung für deren korrekte Funktionalität auf der Seite des Anbieters des Quellsystems liegt.

Nach [TZ16, S. 122] sind auf jedes dieser Subsysteme die Teststufen der Unit-Tests, der Komponententests, der Integrationstests sowie der Systemtests und der Abnahmetests anzuwenden. Auf jeder Teststufe sind Tests für alle acht in ISO-25010 aufgeführten Qualitätskriterien zu entwickeln.

Die einfache Übertragung der Kriterien von klassischen Softwareprodukten auf BI-Systeme ermöglicht ebenfalls eine Übertragung der Metriken zur Bestimmung der Qualität.

Besonders geeignet zur Testung eines BI-Systems scheinen die funktionalen Tests sowie die statistischen Tests zu sein. Funktionale Tests werden verwendet, um zu prüfen, die Funktionalität den Erwartungen entspricht. Mögliche Anwendungsbereiche können Abschnitte des ETL-Prozesses sein, wie etwa die Prüfung, ob die Daten aller Mitarbeiter korrekt vom Data Lake in das Data Warehouse übertragen wurden. Dafür kann ein Abgleich der tatsächlich enthaltenen mit den erwarteten Mitarbeiterdaten erfolgen. Stimmen diese überein, kann von einer korrekt implementierten Logik ausgegangen werden. Ein geeigneter Anwendungsfall für statistische Tests sind beispielsweise die Buchungen der Aufwände der Mitarbeiter. Bei einer verhältnismäßig geringen Schwankung der Anzahl und Stundenzahl der Mitarbeiter ist davon auszugehen, dass auch die Anzahl und die Gesamtzeit der Buchungen sowie die Dauer der einzelnen Buchungen etwa gleichbleibend ist. Durch einen statistischen Test können Abweichungen von den durchschnittlichen Zeiten ermittelt werden. Diese dienen nicht als eindeutige Anzeige für fehlerhafte Vorgänge, bieten aber einen Anhaltspunkt für nicht erwartungsgemäße Vorgänge.

Die Kriterien zur Bestimmung der Qualität eines Business Intelligence-Systems lassen sich nach [LG02] in die Qualität des Systems selbst (*system quality*) und die Qualität der enthaltenen Daten (*information quality*) unterteilen. Für letzteres wurde jedoch ein eigenes Datenqualitätsmodell in der ISO/IEC 25012 entwickelt. [ISO08a] Aufgrund der Wichtigkeit der Qualität der verwendeten Daten wird die Datenqualität als weiteres Qualitätsmerkmal betrachtet, welches getestet werden sollte [TZ16]; eine Betrachtung dieser wird in dieser Arbeit jedoch nicht durchgeführt.

Die Qualität des Systems kann nach Kriterien in drei Bereichen kategorisiert werden: Die Systemfunktionalität, die Benutzbarkeit sowie die vorhandenen Funktionen.

Tabelle 3.2: Qualitätskriterien eines BI-Systems nach [LG02]

Systemfunktionalität	Benutzbarkeit	Funktionen
Performance	Einfache Benutzung	Informationserfassung
Flexibilität	Einfache Erlernbarkeit	Datenbereinigung
Zuverlässigkeit		Datenumformung
Erweiterbarkeit		Statistische Analyse
Wartbarkeit		Simulation
Rechtzeitige Aktualisierungen		Mehrzweckaggregation
Sicherheit		Datenintegration
		Datenkorrektur

Das Testen von Business Intelligence-Systemen bringt jedoch im Vergleich zu Tests herkömmlicher Software einige Herausforderungen mit sich. Der Aufbau eines BI-Systems ist sehr komplex und beinhaltet die unterschiedlichsten Programmiersprachen und Systeme. Nahezu jede dieser Komponenten beinhaltet Daten, deren Qualität entscheidend für das Endergebnis, die Berichterstellung, ist. Diese nehmen somit einen sehr großen Anteil an nötigen Tests ein, da sie zudem über eine komplexe Struktur sowie syntaktische Unterschiede verfügen können, welche ausgeglichen werden müssen, um die Integration der Daten zu ermöglichen. Geschieht die Datenaufbereitung nicht frühzeitig und mit einem Ergebnis in der benötigten Qualität, sind spätere Transformationen nötig, welche zeit- und kostenintensiv sind. Ungleichheiten in den Datenbeständen entstehen etwa durch unterschiedliche Strukturen in den Quellsystemen oder eine bereits im Quellsystem mangelhafte Datenqualität. Dies erfordert eine detaillierte Planung der notwendigen Transformations- und Ladeprozesse. [TZ16]

Zusammenfassend lässt sich sagen, dass die Qualitätssicherung in Business Intelligence-Systemen der Qualitätssicherung in herkömmlicher Software stark ähnelt. Die Besonderheit der Business Intelligence-Systeme liegt darin, dass das Speichern, Transformieren und Lesen von großen Datenmengen im Vordergrund steht. Die Datenqualität stellt jedoch nur eines der neun Kriterien zur Qualitätsbestimmung dar. Die weiteren acht Kriterien lassen sich unabhängig davon testen und wie von [ASD14] bestimmt auf BI-Systeme anwenden. Dafür lassen sich nach [TZ16, S. 122] alle Teststufen verwenden.

## 4 Softwaretests in BI-Systemen

In diesem Kapitel erfolgt die Betrachtung des BI-Systems unterteilt in die Abschnitte *ETL-Prozess*, *Datenspeicher* (Data Warehouse, Data Lake) und *Analyse*. Die Schnittstellen der Quellsysteme werden bei der Betrachtung möglicher Tests nicht berücksichtigt, da die Qualitätssicherung an dieser Stelle im Verantwortungsbereich des Quellsystemanbieters liegt. Im Folgenden beginnt die Ermittlung der Testfälle ab dem Vorliegen der Daten in einer Datei. Der Fokus liegt dabei auf dem Kriterium der Funktionalität.

### 4.1 ETL-Prozess

Funktionale Tests in BI-Systemen sollen sicherstellen, dass die Daten korrekt aus den Quellsystemen geladen, bereinigt, transformiert und in den entsprechenden Speicherort geladen werden. Der Hauptteil dieser Funktionalität wird durch den ETL-Prozess umgesetzt, weshalb diesem im Testbereich die meiste Aufmerksamkeit gilt.

Um einen ETL-Prozess vollständig testen zu können, ist es notwendig, die genauen Anforderungen des spezifischen, zu testenden ETL-Prozesses zu ermitteln. Um es zu ermöglichen, diese Testfälle optimal anzuwenden und auszuwerten, ist ein Prozess nötig, der sich in folgende Schritte unterteilen lässt:

1. Testfälle erstellen:  
Ermitteln der möglichen Testfälle basierend auf den Erkenntnissen aus Schritt 1
2. Tests durchführen:  
Die Daten aus den Quellsystemen laden, Tests durchführen und Auffälligkeiten protokollieren
3. Transformationslogik anwenden:  
Sicherstellen, dass die Daten dem Schema des Data Warehouses entsprechen
4. Daten in das Data Warehouse laden:  
Sicherstellen, dass die Anzahl der erwarteten Datensätze der tatsächlichen Anzahl entspricht und keine ungültigen Werte geladen werden
5. Zusammenfassenden Bericht erstellen:  
Überprüfen von Aufbau, Inhalten, Funktionen und Filtern; enthält für Entscheidungsträger wichtige Daten über den Testprozess
6. Abschluss:  
Abschluss der Testphase

Es gibt folgende Testarten, die durchgeführt werden können, um ETL-Prozesse zu testen:

- Produktvalidierung: Übereinstimmung der Daten in Quellsystemen und Zielsystem

- Anzahl der Datensätze: Übereinstimmung der Anzahl der Datensätze in Quellsystemen und Zielsystem
- Vollständigkeit der Daten: Überprüfung, ob bei der Übertragung von Quell- zu Zielsystem alle Daten erhalten geblieben sind
- Metadaten: Überprüfung der Korrektheit von Datentypen, -länge, -indizes und Einschränkungen
- Performanz: Sicherstellung einer angemessenen Dauer für die Übertragung der Daten von Quell- zu Zielsystem
- Datenqualität: Gültigkeit der Daten in Bezug auf Syntax (ungültige Zeichen, Groß- und Kleinschreibung) und Referenzen (Zahl, Datum, Nullprüfung) untersuchen
- Datenintegration: Sicherstellung der korrekten Integration der Daten aus verschiedenen Quellen sowie die Einhaltung der Schwellwerte
- Berichte: Prüfung von Layout und Funktionalität
- Benutzerakzeptanz: Sicherstellung der korrekten Funktionsweise aus Nutzersicht
- GUI: Sicherstellung der korrekten Funktionsweise der grafischen Benutzeroberfläche
- Anwendungsmigration: Sicherstellung der korrekten Funktionsweise der Anwendung auf anderen Plattformen

nach [Talb]

Aus dieser Menge an Testarten ergibt sich eine Vielzahl an unterschiedlichen möglichen Testfällen, welche im Folgenden ausgeführt sind.

Der Aufbau der Testfälle gliedert sich in eine Testnummer sowie eine kurze Bezeichnung des Testfalles zur eindeutigen Identifizierung der Testfälle; eine Beschreibung, die den Nutzen des Testfalles darstellt; den benötigten Ausgangszustand, um den Testfall unter den optimalen Bedingungen durchzuführen; den geplanten Testablauf, um den Test korrekt durchzuführen; die nötigen Prüfungen und das erwartete Ergebnis, um den Erfolg des Tests sicherzustellen sowie Bemerkungen über Besonderheiten des Testfalles, wie etwa die Durchführbarkeit in unterschiedlichen Abschnitten des BI-Systems.



**Datensätze hinzufügen**

Tabelle 4.1: Testfall 1.01: Datensätzen in eine leere Datenbank laden

<b>Bezeichnung</b>	Testfall 1.01: Datensätzen in eine leere Datenbank laden
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Es werden Datensätze in eine leere Datenbank eingefügt, um das Hinzufügen ohne den Einfluss bestehender Datensätze zu testen.
<b>Ausgangszustand</b>	Datenbank: - enthält die gewünschte Struktur - enthält keine Datensätze  Datei: - enthält vollständige, syntaktisch korrekte Datensätze
<b>Ablauf</b>	1. Datei, die hinzuzufügende Datensätze enthält, in das BI-System laden 2. ETL-Prozess starten 3. Abgleich der Datensätze aus der Datei und der Datenbank
<b>Prüfungen</b>	Die Anzahl der Datensätze in der Datei und in der Datenbank stimmen überein. Jeder Datensatz wurde vollständig in die Datenbank übertragen.
<b>Erwartetes Ergebnis</b>	Die eingefügten Datensätze sind vollständig und syntaktisch korrekt in der Datenbank enthalten.
<b>Bemerkungen</b>	Je nach Aufbau des Data Warehouses kann mit diesem Test das Laden der Daten aus dem Quellsystem oder aus dem Data Lake in die Basisdatenbank oder, je nach gewünschten Transformationen, auch in die Ableitungsdatenbank, überprüft werden.

Tabelle 4.2: Testfall 1.02: Hinzufügen einer leeren Datei

<b>Bezeichnung</b>	Testfall 1.02: Hinzufügen einer leeren Datei
<b>Testkategorie</b>	Negativtest
<b>Beschreibung</b>	Es wird eine leere Datei hinzugefügt, um zu prüfen, dass keine Änderung am Inhalt der Datenbank erfolgt.

<b>Ausgangszustand</b>	<p>Datenbank:</p> <ul style="list-style-type: none"> <li>- enthält die gewünschte Struktur</li> <li>- kann leer sein oder Datensätze enthalten</li> </ul> <p>Datei:</p> <ul style="list-style-type: none"> <li>- enthält keine Datensätze</li> </ul>
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Leere Datei in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich der Anzahl der Datensätze vor und nach dem Starten des ETL-Prozesses</li> </ol>
<b>Prüfungen</b>	Die Anzahl der Datensätze in der Datenbank bleibt unverändert.
<b>Erwartetes Ergebnis</b>	Der Zustand der Datenbank bleibt unverändert.
<b>Bemerkungen</b>	Je nach Spezifikation ist ein Abbruch des ETL-Prozesses oder die Durchführung ohne Änderungen in der Datenbank möglich.

Tabelle 4.3: Testfall 1.03: Wiederholtes Hinzufügen einer bereits verarbeiteten Datei

<b>Bezeichnung</b>	Testfall 1.03: Wiederholtes Hinzufügen einer bereits verarbeiteten Datei
<b>Testkategorie</b>	Negativtest
<b>Beschreibung</b>	Eine Datei, welche bereits im BI-System verarbeitet wurde, wird nochmals hinzugefügt, um zu prüfen, dass dadurch keine Änderungen in der Datenbank erfolgen.
<b>Ausgangszustand</b>	<p>Datenbank:</p> <ul style="list-style-type: none"> <li>- enthält die gewünschte Struktur</li> <li>- enthält Datensätze</li> </ul> <p>Datei:</p> <ul style="list-style-type: none"> <li>- enthält Datensätze, die sich bereits in der Datenbank befinden</li> </ul>
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Datei in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich der Anzahl der Datensätze vor und nach dem Starten des ETL-Prozesses</li> </ol>
<b>Prüfungen</b>	Die Anzahl und Werte der Datensätze in der Datenbank bleiben unverändert.

<b>Erwartetes Ergebnis</b>	Der Zustand der Datenbank bleibt unverändert.
<b>Bemerkungen</b>	Je nach Spezifikation ist ein Abbruch des ETL-Prozesses oder die Durchführung ohne Änderungen in der Datenbank möglich.

Tabelle 4.4: Testfall 1.04: Zeitgleiches Hinzufügen mehrerer unterschiedlicher Dateien

<b>Bezeichnung</b>	Testfall 1.04: Zeitgleiches Hinzufügen mehrerer unterschiedlicher Dateien
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Es werden mehrere unterschiedliche Dateien zur gleichen Zeit hinzugefügt, um zu prüfen, dass die Dateien in der Reihenfolge des Hinzufügens abgearbeitet werden.
<b>Ausgangszustand</b>	<p>Datenbank:</p> <ul style="list-style-type: none"> <li>- enthält die gewünschte Struktur</li> <li>- enthält keine Datensätze</li> </ul> <p>Dateien:</p> <ul style="list-style-type: none"> <li>- enthalten unterschiedliche Datensätze</li> </ul>
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Dateien in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich der Anzahl der Datensätze in den Dateien und der Datenbank unter Berücksichtigung der Reihenfolge</li> </ol>
<b>Prüfungen</b>	<p>Die Anzahl der Datensätze in den Dateien und der Datenbank stimmt überein.</p> <p>Die Reihenfolge der Datensätze in den Dateien und der Datenbank stimmt überein.</p> <p>Mögliche Änderungen in den Datensätzen wurden korrekt übertragen.</p>
<b>Erwartetes Ergebnis</b>	Die Datenbank enthält alle Datensätze in der korrekten Reihenfolge. Mögliche Änderungen wurden korrekt umgesetzt.

Tabelle 4.5: Testfall 1.05: Hinzufügen einer fehlerhaften Datei

<b>Bezeichnung</b>	Testfall 1.05: Hinzufügen einer fehlerhaften Datei
<b>Testkategorie</b>	Negativtest
<b>Beschreibung</b>	Es wird eine fehlerhafte Datei hinzugefügt, um zu prüfen, dass die Datenbank wieder in den Ausgangszustand versetzt wird.
<b>Ausgangszustand</b>	<p>Datenbank:</p> <ul style="list-style-type: none"> <li>- enthält die gewünschte Struktur</li> <li>- kann leer sein oder Datensätze enthalten</li> </ul> <p>Datei:</p> <ul style="list-style-type: none"> <li>- enthält mehrere Datensätze</li> </ul>
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Datei in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich der Anzahl der Datensätze in der Datenbank vor und nach dem Starten des ETL-Prozesses</li> </ol>
<b>Prüfungen</b>	Der ETL-Prozess bricht ab. Die Anzahl der Datensätze in der Datenbank bleibt unverändert.
<b>Erwartetes Ergebnis</b>	Der Zustand der Datenbank bleibt unverändert.

Tabelle 4.6: Testfall 1.06: Hinzufügen einer Datei mit fehlerhaften Datensätzen

<b>Bezeichnung</b>	Testfall 1.06: Hinzufügen einer Datei mit fehlerhaften Datensätzen
<b>Testkategorie</b>	Negativtest
<b>Beschreibung</b>	Es wird eine Datei hinzugefügt, die fehlerhafte Datensätze enthält, um zu prüfen, dass die Datenbank wieder in den Ausgangszustand versetzt wird.
<b>Ausgangszustand</b>	<p>Datenbank:</p> <ul style="list-style-type: none"> <li>- enthält die gewünschte Struktur</li> <li>- kann leer sein oder Datensätze enthalten</li> </ul> <p>Datei:</p> <ul style="list-style-type: none"> <li>- enthält mehrere Datensätze, von denen mindestens einer fehlerhafte Werte enthält</li> </ul>

<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Datei in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich der Anzahl der Datensätze vor und nach dem Starten des ETL-Prozesses</li> </ol>
<b>Prüfungen</b>	Die Anzahl der Datensätze in der Datenbank bleibt unverändert.
<b>Erwartetes Ergebnis</b>	Der Zustand der Datenbank bleibt unverändert.

Tabelle 4.7: Testfall 1.07: Hinzufügen einer Datei mit veränderter Datenstruktur

<b>Bezeichnung</b>	Testfall 1.07: Hinzufügen einer Datei mit veränderter Datenstruktur
<b>Testkategorie</b>	Negativtest
<b>Beschreibung</b>	Es wird eine Datei hinzugefügt, die Datensätze enthält, deren Datenstruktur sich von der der Tabellen in der Datenbank unterscheidet.
<b>Ausgangszustand</b>	<p>Datenbank:</p> <ul style="list-style-type: none"> <li>- enthält die gewünschte Struktur</li> <li>- kann leer sein oder Datensätze enthalten</li> </ul> <p>Datei:</p> <ul style="list-style-type: none"> <li>- enthält Datensätze mit anderer Struktur als die entsprechende Datenbanktabelle</li> </ul>
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Datei in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich der Anzahl der Datensätze vor und nach dem Starten des ETL-Prozesses</li> </ol>
<b>Prüfungen</b>	Die Anzahl der Datensätze in der Datenbank bleibt unverändert.
<b>Erwartetes Ergebnis</b>	Der Zustand der Datenbank bleibt unverändert.
<b>Bemerkungen</b>	Je nach Spezifikation ist eine automatische Durchführung von Transformationen möglich, was eine fehlerlose Durchführung dieses Testfalles ermöglicht.

**Datensätze aktualisieren**

Tabelle 4.8: Testfall 1.08: Slowly Changing Dimension Typ 0

<b>Bezeichnung</b>	Testfall 1.08: Slowly Changing Dimension Typ 0
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	In einer Datenbank mit einem bestehenden Datensatz werden Änderungen an dem Wert eines Attributes vorgenommen, um zu prüfen, dass dieser Wert nicht geändert wird.
<b>Ausgangszustand</b>	Datenbank: - enthält die gewünschte Struktur - enthält einen Datensatz  Datei: - enthält eine leicht veränderte Version des Datensatzes, der sich bereits in der Datenbank befindet
<b>Ablauf</b>	1. Datei mit dem veränderten Datensatz in das BI-System laden 2. ETL-Prozess starten 3. Abgleich des Datensatzes mit seiner bisherigen Version
<b>Prüfungen</b>	Der Datensatz enthält weiterhin die ursprünglichen Werte.
<b>Erwartetes Ergebnis</b>	Es wurden keine Änderungen an dem Datensatz vorgenommen.

Tabelle 4.9: Testfall 1.09: Slowly Changing Dimension Typ 1

<b>Bezeichnung</b>	Testfall 1.09: Slowly Changing Dimension Typ 1
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	In einer Datenbank mit einem bestehenden Datensatz werden Änderungen an dem Wert eines Attributes vorgenommen, um zu prüfen, dass dieser Wert überschrieben wird.
<b>Ausgangszustand</b>	Datenbank: - enthält die gewünschte Struktur - enthält einen Datensatz  Datei: - enthält eine leicht veränderte Version des Datensatzes, der sich bereits in der Datenbank befindet

<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Datei mit dem veränderten Datensatz in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich des Datensatzes mit seiner bisherigen Version</li> </ol>
<b>Prüfungen</b>	Der veränderte Wert ist im Datensatz enthalten.
<b>Erwartetes Ergebnis</b>	Der Wert des Attributes wurde wie gewünscht geändert.

Tabelle 4.10: Testfall 1.10: Slowly Changing Dimension Typ 2

<b>Bezeichnung</b>	Testfall 1.10: Slowly Changing Dimension Typ 2
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	In einer Datenbank mit einem bestehenden Datensatz werden Änderungen an einem Attribut vorgenommen, um zu prüfen, dass ein neuer, gültiger Datensatz für dieses Realweltobjekt angelegt wird und der bisherige Datensatz als ungültig markiert wird.
<b>Ausgangszustand</b>	<p>Datenbank:</p> <ul style="list-style-type: none"> <li>- enthält die gewünschte Struktur</li> <li>- enthält einen Datensatz</li> </ul> <p>Datei:</p> <ul style="list-style-type: none"> <li>- enthält eine leicht veränderte Version des Datensatzes, der sich bereits in der Datenbank befindet</li> </ul>
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Datei mit dem veränderten Datensatz in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich des Datensatzes mit seiner bisherigen Version</li> </ol>
<b>Prüfungen</b>	<p>Bisher gültiger Datensatz:</p> <ul style="list-style-type: none"> <li>- enthält ein Datum für das Ende der Gültigkeit, das in der Vergangenheit liegt</li> <li>- Attribut zur Markierung der Gültigkeit wurde auf <i>ungültig</i> gesetzt</li> </ul> <p>Nun gültiger Datensatz:</p> <ul style="list-style-type: none"> <li>- wurde neu erstellt</li> <li>- Attribut zur Markierung der Gültigkeit wurde auf <i>gültig</i> gesetzt</li> <li>- Datum für den Anfang der Gültigkeit schließt lückenlos an den bisher gültigen Datensatz an</li> </ul>

<b>Erwartetes Ergebnis</b>	Die Datenbank enthält den bisherigen, nun ungültigen Datensatz sowie einen neuen, gültigen Datensatz mit den gewünschten Änderungen.
----------------------------	--

Tabelle 4.11: Testfall 1.11: Slowly Changing Dimension Typ 3

<b>Bezeichnung</b>	Testfall 1.11: Slowly Changing Dimension Typ 3
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	In einer Datenbank mit einem bestehenden Datensatz werden Änderungen an einem Attribut vorgenommen, um zu prüfen, dass der neue Wert eines Attributes zusätzlich zu dem vorherigen Wert im Datensatz enthalten ist.
<b>Ausgangszustand</b>	<p>Datenbank:</p> <ul style="list-style-type: none"> <li>- enthält die gewünschte Struktur</li> <li>- enthält einen Datensatz</li> </ul> <p>Datei:</p> <ul style="list-style-type: none"> <li>- enthält eine leicht veränderte Version des Datensatzes, der sich bereits in der Datenbank befindet</li> </ul>
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Datei mit dem veränderten Datensatz in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich des Datensatzes mit seiner bisherigen Version</li> </ol>
<b>Prüfungen</b>	<p>Der Datensatz enthält den bisherigen Wert des Attributes in einer extra Spalte für vorherige Werte.</p> <p>Der Datensatz enthält den aktualisierten Wert des Attributes in der Spalte für die aktuellen Attributwerte.</p>
<b>Erwartetes Ergebnis</b>	Der Datensatz enthält sowohl den bisherigen als auch den aktualisierten Wert des Attributes.

Tabelle 4.12: Testfall 1.12: Slowly Changing Dimension Typ 4

<b>Bezeichnung</b>	Testfall 1.12: Slowly Changing Dimension Typ 4
<b>Testkategorie</b>	Positivtest



<b>Beschreibung</b>	In einer Datenbank mit einem bestehenden Datensatz werden Änderungen an einem Attribut vorgenommen, um zu prüfen, dass in einer weiteren Tabelle ein Datensatz angelegt wird, der den bisherigen Attributwert enthält.
<b>Ausgangszustand</b>	<p>Datenbank:</p> <ul style="list-style-type: none"> <li>- enthält zwei Tabellen in der gewünschten Struktur</li> <li>- die Historisierungstabelle enthält Spalten für das Datum von Beginn und Ende der Gültigkeit</li> <li>- die Historisierungstabelle enthält keine Datensätze</li> <li>- die Haupttabelle enthält einen Datensatz</li> </ul> <p>Datei:</p> <ul style="list-style-type: none"> <li>- enthält eine leicht veränderte Version des Datensatzes, der sich bereits in der Datenbank befindet</li> </ul>
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Datei mit dem veränderten Datensatz in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich des Datensatzes mit seiner bisherigen Version</li> </ol>
<b>Prüfungen</b>	<p>Die Haupttabelle enthält weiterhin nur einen Datensatz, der den aktuellen Wert enthält.</p> <p>Die Historisierungstabelle enthält einen Datensatz, welcher den bisherigen Wert des Attributes sowie einen in der Vergangenheit liegenden Gültigkeitszeitraum enthält.</p>
<b>Erwartetes Ergebnis</b>	Der Datensatz in der Haupttabelle wurde aktualisiert. In der Historisierungstabelle wurde ein historisierter Datensatz angelegt.

Tabelle 4.13: Testfall 1.13: Slowly Changing Dimension Typ 6

<b>Bezeichnung</b>	Testfall 1.13: Slowly Changing Dimension Typ 6
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	In einer Datenbank mit einem bestehenden Datensatz werden Änderungen an einem Attribut vorgenommen, um zu prüfen, dass der bisher gültige Datensatz das neue Attribut erhält und als ungültig markiert wird sowie ein neuer, gültiger Datensatz angelegt wird.

<b>Ausgangszustand</b>	<p>Datenbank:</p> <ul style="list-style-type: none"> <li>- enthält die gewünschte Struktur</li> <li>- enthält einen Datensatz</li> </ul> <p>Datei:</p> <ul style="list-style-type: none"> <li>- enthält eine leicht veränderte Version des Datensatzes, der sich bereits in der Datenbank befindet</li> </ul>
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Datei mit dem veränderten Datensatz in das BI-System laden</li> <li>2. ETL-Prozess starten</li> <li>3. Abgleich des Datensatzes mit seiner bisherigen Version</li> </ol>
<b>Prüfungen</b>	<p>Der bisherige Datensatz enthält den aktualisierten Wert in der Spalte für den aktuellen Attributwert.</p> <p>Der bisherige Datensatz enthält ein in der Vergangenheit liegendes Datum für das Ende des Gültigkeitszeitraumes.</p> <p>Der bisherige Datensatz ist als ungültig markiert.</p> <p>Es wurde ein neuer, gültiger Datensatz angelegt.</p> <p>Der neue Datensatz enthält den aktuellen Attributwert.</p> <p>Der neue Datensatz hat einen aktuellen Gültigkeitszeitraum und ist als gültig markiert.</p>
<b>Erwartetes Ergebnis</b>	<p>Der bisherige Datensatz wurde als ungültig markiert. Es wurde ein neuer, gültiger Datensatz angelegt.</p>

Das Testen von ETL-Prozessen bringt Herausforderungen mit sich. Diese bestehen aus der Behandlung von Daten, die bei der Übertragung verloren gegangen sind, beschädigt wurden, nur begrenzt verfügbar oder mehrfach vorhanden sind; aus der ungenügenden Betrachtung von Anforderungen, die wesentlich für die korrekte Erfüllung der Ansprüche an Tests notwendig sind; die Verwendung einer instabilen Testumgebung sowie die Verwendung von veralteten ETL-Test-Anwendungen. [Talb]

Es existieren verschiedene Anwendungen, welche auf das Erstellen und Testen von ETL-Prozessen ausgelegt sind. Einige dieser Anwendungen können zusätzlich cloud-basierte Architekturen testen; einige wurden speziell für diese entwickelt. Bekannte Anbieter von Business Intelligence-Systemen mit Unterstützung cloud-basierter Architektur sind beispielsweise Google (*Google Big Query [Goo]*), Amazon (*Amazon Web Services [Ama]*) oder Microsoft (*Microsoft Azure [Mic]*). Existierende Anwendungen für das Erstellen eines Business Intelligence-Systems, bei denen die Möglichkeit des Testens dieser erstellten Prozesse integriert ist, wurden beispielsweise von talend [Tala], RightData [Rig] und QuerySurge [Que] entwickelt.

Bei der Auswahl eines geeigneten Tools für die Erstellung und Testung eines ETL-Prozesses gibt es verschiedene Kriterien, die entsprechend den individuellen spezifischen Anforderungen nach beachtet werden sollten:

- Grafische Oberfläche, um die Erstellung des ETL-Prozesses zu erleichtern

- Automatische Code-Generierung, um eine schnellere Entwicklung mit geringerer Fehleranzahl zu ermöglichen
- Integrierte Datenkonnektoren, um den Zugriff auf Daten in verschiedenen Formaten, beispielsweise als Datei, in einer Datenbank, in einem Anwendungspaket, in einem weiteren BI-System zu gewähren
- Content-Management-Funktionen, die einen Wechsel zwischen Entwicklungs-, Test- und Produktionsumgebung ermöglichen
- Debugging-Möglichkeiten, die den Datenfluss in Echtzeit nachverfolgbar machen und Berichte über das Verhalten jeder einzelnen Zeile geben

## 4.2 Datenspeicher

### 4.2.1 Data Warehouse

Da ein Data Warehouse aus Datenbanken besteht, sind diese der Ansatz, nach dem ein Data Warehouse zu testen ist. Die Bestandteile einer Datenbank, die getestet werden können, sind Transaktionen, das Datenbankschema, Trigger, gespeicherte Prozeduren sowie Feldbeschränkungen. [mysb] Da diese Teil des gesamten BI-Systems sind, ist keine eindeutige Trennung der Bereiche möglich.

Zu den Transaktionen gehören das Lesen von Datensätzen aus der Datenbank, das Aktualisieren von Datensätzen, das Hinzufügen von Datensätzen, das Löschen von Datensätzen sowie dazugehörige Rollback-Anweisungen, die der Wiederherstellung des Ausgangszustandes der Datenbank nach gescheiterten Transaktionen dienen.

Das Datenbankschema lässt sich durch das Vorhandensein von Primär- und Fremdschlüsseln, der vorgabengemäßen Benennung von Attributen sowie der Sicherstellung, dass enthaltene Werte den Anforderungen entsprechen, überprüfen.

Die Trigger einer Datenbank sind Ereignisse, welche bestimmte Transaktionen auslösen. Getestet wird dabei, ob nach dem Auslösen eines Triggers die gewünschte Transaktion durchgeführt wird. Die erfolgreiche Durchführung ist dabei durch die Überprüfung des Datenbankinhaltes möglich. Datenbankprozeduren bestehen aus einer Abfolge von Transaktionen, welche nach Start der Prozedur in der festgelegten Reihenfolge ausgeführt werden. Der Test der korrekten Ausführung erfolgt durch die Kontrolle, ob der Datenbankzustand dem erwarteten Zustand entspricht.

Unter Feldbeschränkungen sind die möglichen Werte für Attribute, festgelegte Standardwerte oder Beschränkungen der Länge des Wertes zusammengefasst. [mysb]

Die Herausforderungen, die beim Testen auftreten können, beziehen sich vorrangig auf die verwendeten Daten. Diese können mehrfach vorkommen, durch einen noch nicht ausreichend getesteten ETL-Prozess verändert werden, einen zu großen Umfang bzw. Komplexität aufweisen oder von Beginn an nicht korrekt erfasst wurden sein. Weitere mögliche Herausforderungen können fehlende Berechtigungen zum Starten nötiger ETL-Prozesse, nicht ausreichend oder fehlerhaft definierte Geschäftsprozesse oder eine instabile Testumgebung darstellen. [mysa]

Tabelle 4.14: Testfall 2.01: Alle Pflichtattribute enthalten einen Wert

<b>Bezeichnung</b>	Testfall 2.01: Alle Pflichtattribute enthalten einen Wert
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Alle Datensätze werden darauf untersucht, ob die Pflichtattribute einen Wert enthalten.
<b>Ausgangszustand</b>	Datenbank: - enthält die gewünschte Struktur - enthält mehrere Datensätze
<b>Ablauf</b>	Abfrage aller NULL-Werte in den Pflichtattributen aller Datensätze
<b>Prüfungen</b>	Die Abfrage sollte keinen enthaltenen NULL-Wert zurückgeben.
<b>Erwartetes Ergebnis</b>	Es wird kein NULL-Wert gefunden.
<b>Anmerkungen</b>	Dieser Test ist für bereits bestehende, befüllte Datenbanken zu empfehlen, bei denen während dem Hinzufügen keine Überprüfungen vorgenommen wurden. Bei zukünftigen hinzuzufügenden Datensätzen sollte die Überprüfung bereits während des Hinzufügens stattfinden.

Tabelle 4.15: Testfall 2.02: Alle Werte entsprechen den syntaktischen Anforderungen

<b>Bezeichnung</b>	Testfall 2.02: Alle Werte entsprechen den syntaktischen Anforderungen
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Alle Werte jedes Datensatzes werden darauf untersucht, ob diese den syntaktischen Anforderungen entsprechen.
<b>Ausgangszustand</b>	Datenbank: - enthält die gewünschte Struktur - enthält mehrere Datensätze
<b>Ablauf</b>	1. Laden aller Datensätze aus der Datenbank 2. Überprüfung der Werte entsprechend den Anforderungen
<b>Prüfungen</b>	Jeder Wert entspricht den Anforderungen.

<b>Erwartetes Ergebnis</b>	Es wird kein Wert angezeigt, der nicht den Anforderungen entspricht.
<b>Anmerkungen</b>	Die Anforderungen können den Zeichentyp, die Zeichenlänge sowie Einschränkungen in den erlaubten Werten umfassen.

Tabelle 4.16: Testfall 2.03: Grenzwerte werden korrekt berücksichtigt

<b>Bezeichnung</b>	Testfall 2.03: Grenzwerte werden korrekt berücksichtigt
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Jeder Wert liegt innerhalb der festgelegten Grenzwerte.
<b>Ausgangszustand</b>	Datenbank: - enthält die gewünschte Struktur - enthält mehrere Datensätze
<b>Ablauf</b>	1. Laden aller Datensätze aus der Datenbank 2. Überprüfung der Werte, für die Grenzwerte vorhanden sind
<b>Prüfungen</b>	Jeder Wert liegt innerhalb der Grenzwerte.
<b>Erwartetes Ergebnis</b>	Es wird kein Wert angezeigt, der nicht in den Grenzwerten liegt.

Tabelle 4.17: Testfall 2.04: Berechnete Attribute enthalten die korrekten Formeln

<b>Bezeichnung</b>	Testfall 2.04: Berechnete Attribute enthalten die korrekten Formeln
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Jedes berechnete Attribut enthält die korrekte Formel.
<b>Ausgangszustand</b>	Datenbank: - enthält die gewünschte Struktur - enthält mehrere Datensätze
<b>Ablauf</b>	1. Laden der Tabellenstruktur 2. Abgleich der Formeln aus der Tabellenstruktur mit den in der Anforderung festgelegten Formeln

<b>Prüfungen</b>	Korrektheit der Formel jedes Attributes
<b>Erwartetes Ergebnis</b>	Jede Formel ist korrekt in der Tabelle hinterlegt.

### 4.2.2 Data Lake

Da ein Data Lake hauptsächlich als Ablage für Dateien aller Art dient (siehe Kapitel 2.4.2), existiert nur eine geringe Anzahl an möglichen Testfällen. Welche Testfälle für einen spezifischen Data Lake sinnvoll sind, ist abhängig von dessen Struktur. Werden die Daten in einer Ordnerstruktur abgelegt, ist ein Test für die automatische Speicherung am gewünschten Ablageort möglich. Verfügt der Data Lake über eine Archivzone, in welcher bereits verarbeitete Daten abgelegt werden, kann die korrekte Ablage sowie das Entfernen am bisherigen Speicherort getestet werden.

Tabelle 4.18: Testfall 2.05: Datei wird korrekt in gewünschter Struktur abgelegt

<b>Bezeichnung</b>	Testfall 2.05: Datei wird korrekt in gewünschter Struktur abgelegt
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Die Datei wird in einer festgelegten Struktur im Data Lake abgelegt.
<b>Ausgangszustand</b>	Data Lake: - enthält eine Ordnerstruktur
<b>Ablauf</b>	1. Hinzufügen der Datei in den Data Lake 2. Überprüfen des Speicherortes
<b>Prüfungen</b>	Die Datei liegt an der erwarteten Stelle im Data Lake.
<b>Erwartetes Ergebnis</b>	Die Datei wurde korrekt in der Struktur des Data Lakes abgelegt.
<b>Anmerkungen</b>	Ein Data Lake muss nicht zwingend über eine Ordnerstruktur verfügen. In diesem Fall ist nur zu prüfen, ob die Datei im Data Lake enthalten.

Tabelle 4.19: Testfall 2.06: Datei wird nach Speichern in der Datenbank in Archivzone abgelegt

<b>Bezeichnung</b>	Testfall 2.06: Datei wird nach Speichern in der Datenbank in Archivzone abgelegt
--------------------	--

<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Die Datei wird von einer Stelle des Data Lakes an eine andere Stelle in einer festgelegten Struktur verschoben. Die Datei wird an der vorherigen Stelle gelöscht.
<b>Ausgangszustand</b>	Data Lake: - enthält eine Ordnerstruktur
<b>Ablauf</b>	1. Hinzufügen der Datei in den Data Lake 2. Überprüfen des Speicherortes
<b>Prüfungen</b>	Die Datei liegt an der erwarteten Stelle im Data Lake. Die Datei wurde vom vorherigen Speicherort entfernt.
<b>Erwartetes Ergebnis</b>	Die Datei wurde vom bisherigen Speicherort entfernt und an der korrekten Stelle abgelegt.

### 4.3 Analyse-Anwendungen

Die verwendete Analyse-Anwendung ist kein direkter Bestandteil des BI-Systems, weshalb kein Zugriff auf deren Code besteht. Die Funktionalität der Analyse-Anwendung wird durch den Anbieter dessen getestet. Somit sind keine Tests direkt für die Analyse-Anwendung durchführbar. An dieser Stelle kann jedoch getestet werden, dass die an die Analyse-Anwendung übertragenen Daten korrekt verarbeitet werden. Zu der Verarbeitung dieser Daten gehört die Berechnung von Werten, welche im Bericht angezeigt werden. Die Formeln für die Berechnung dieser sollten getestet werden, ebenso die Darstellung der Ergebnisse im korrekten Format. Wird ein Bericht erstellt, welcher die Möglichkeit bietet, die angezeigten Daten nach bestimmten Kriterien zu filtern, kann das Vorhandensein aller gewünschten Filter sowie deren korrekte Anwendung auf die Daten getestet werden.

Tabelle 4.20: Testfall 3.01: Filter sind vollständig im Bericht enthalten

<b>Bezeichnung</b>	Testfall 3.01: Filter sind vollständig im Bericht enthalten
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Alle erwarteten Filter sind in den Bericht integriert.
<b>Ausgangszustand</b>	Es existiert ein vollständiger Bericht.
<b>Ablauf</b>	1. Ermitteln der verfügbaren Filter 2. Abgleich mit den erforderlichen Filtern aus den Anforderungen
<b>Prüfungen</b>	Vorhandensein aller erwarteten Filter

<b>Erwartetes Ergebnis</b>	Alle erforderlichen Filter sind im Bericht enthalten.
----------------------------	---

Tabelle 4.21: Testfall 3.02: Korrekte Filterung aller Werte

<b>Bezeichnung</b>	Testfall 3.02: Korrekte Filterung aller Werte
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Ein Filter, der in einem Bericht angewendet wird, filtert alle erwünschten Daten.
<b>Ausgangszustand</b>	Es existiert ein vollständiger Bericht.
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Auswahl eines Filters</li> <li>2. Prüfung der enthaltenen Daten auf die Anwendung des Filters</li> </ol>
<b>Prüfungen</b>	Die Daten wurden nach dem ausgewählten Filter gefiltert.
<b>Erwartetes Ergebnis</b>	Die Anwendung des Filters bewirkt die Filterung der Daten, die im Bericht enthalten sind.

Tabelle 4.22: Testfall 3.03: Darstellung aller Werte im korrekten Format

<b>Bezeichnung</b>	Testfall 3.03: Darstellung aller Werte im korrekten Format
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Alle im Bericht enthaltenen Werte sind im korrekten Format dargestellt.
<b>Ausgangszustand</b>	Es existiert ein vollständiger Bericht.
<b>Ablauf</b>	<ol style="list-style-type: none"> <li>1. Ermittlung aller enthaltenen Werte</li> <li>2. Abgleich des Formates der ermittelten Werte mit dem in den Anforderungen spezifizierten Format</li> </ol>
<b>Prüfungen</b>	Das Format eines Wertes entspricht der Spezifikation.
<b>Erwartetes Ergebnis</b>	Alle Werten sind im erwarteten Format dargestellt.



Tabelle 4.23: Testfall 3.04: Berechnungen erfolgen auf Basis der korrekten Formeln

<b>Bezeichnung</b>	Testfall 3.04: Berechnungen erfolgen auf Basis der korrekten Formeln
<b>Testkategorie</b>	Positivtest
<b>Beschreibung</b>	Alle im Bericht enthaltenen Werte sind mit den korrekten Formeln berechnet.
<b>Ausgangszustand</b>	Es existiert ein vollständiger Bericht.
<b>Ablauf</b>	<ol style="list-style-type: none"><li>1. Ermittlung aller verwendeten Formeln</li><li>2. Abgleich der verwendeten Formeln mit den in den Anforderungen spezifizierten Formeln</li></ol>
<b>Prüfungen</b>	Die Formel für die Berechnung des Wertes ist korrekt.
<b>Erwartetes Ergebnis</b>	Alle Werte enthalten die korrekten Formeln.

## 5 Prototypische Umsetzung geeigneter Testverfahren

### 5.1 Aufbau des Testsystems

Im Rahmen dieser Arbeit wird ein Softwaretest umgesetzt, um zu prüfen, ob sich die Überlegungen aus dem vorherigen Kapitel auf ein real existierendes Business Intelligence-System anwenden lassen. Um den Aufbau des Testes nachvollziehbar zu machen, wird im Folgenden der Aufbau des Testsystems erläutert.

Bei dem Testsystem handelt es sich um ein Cloud-BI-System, welches auf Microsoft Azure basiert. Damit lässt sich ein wie in Kapitel 2 beschriebenes System umsetzen. Der konkrete Aufbau des Systems ist in Abbildung 5.1 dargestellt.



Abbildung 5.1: Architektur des Testsystems

Das Laden der Daten aus den Quellsystemen erfolgt täglich. Mit Hilfe einer *Azure Logic App* werden die Daten in Form von XML-Dateien in den Data Lake importiert. Dieser ist in eine *Landing Zone* sowie eine *Archive Zone* unterteilt. Die Dateien werden zu Beginn in die Landing Zone geladen. Sowohl die Landing Zone als auch die Archive Zone enthalten eine Ordnerstruktur, welche eine Sortierung der Dateien nach dem Datum ermöglicht. Danach wird die Reihenfolge der Verarbeitung der Dateien festgelegt.

Anschließend werden die Daten durch eine weitere Logic App aus der Landing Zone des Data Lakes in die *Staging Area* (enthält die Basisdatenbank) des Data Warehouses geladen. Von dort aus erfolgen durch die selbe Logic App die Anpassung und die Übertragung der Daten in die *Target Area* (enthält die Ableitungsdatenbank). Diese Logic App enthält SQL-Prozeduren, welche dazu dienen, die Daten in die Data Warehouse-Tabellen einzusortieren sowie die Datensätze auf Vollständigkeit zu überprüfen.

Die Aufgabe der Data Marts (Analysedatenbanken) wird durch den *Azure Analysis Service* übernommen. Dort erfolgt die Berechnung der Kennzahlen.

Durch eine weitere Logic App erfolgt das Laden vom Data Lake in das Data Warehouse. Diese beinhaltet eine SQL-Prozedur, deren Aufgabe darin besteht, die Datensätze der Datei aus dem

Lake passend in die Tabellen des Data Warehouses einzusetzen - zuerst in die Staging-Tabelle, anschließend in die Target-Tabelle. Außerdem wird das Verschieben der Datei in die Archive Zone des Data Lakes durchgeführt.

Die Verwendung der Microsoft Azure Cloud bringt einige Besonderheiten im Vergleich zur klassischen Softwareentwicklung und zu selbst programmierten BI-Systemen mit sich. Sowohl in der klassischen Softwareentwicklung als auch in selbst programmierten BI-Systemen erfolgt die Entwicklung der Programmlogik in einer bestimmten Programmiersprache wie beispielsweise Java. Das Testen dieser Logik erfolgt für gewöhnlich durch Tests, die in der selben Programmiersprache wie die Logik geschrieben wurden. Zur Unterstützung wurden für viele Programmiersprachen Frameworks entwickelt, welche Funktionen enthalten, die den Testprozess vereinfachen. Ein Beispiel für ein Framework, welches verwendet werden kann, wenn die Tests in Java geschrieben wurden, stellt JUnit dar.

Die Entwicklung eines BI-Systems unter Verwendung von Microsoft Azure erfolgt mit Hilfe einer grafischen Benutzeroberfläche und folgt dem sogenannten *Low-Code-Ansatz*. Das bedeutet, dass der Entwickler nur einen geringen Anteil der Entwicklung mit Hilfe einer Programmiersprache umsetzt. Somit lässt sich nur ein geringer Anteil des BI-Systems über die eben beschriebene Vorgehensweise testen. Dazu gehören die Berechnung der Kennzahlen sowie die Datenbank und darin existierende Prozeduren. Um mit dem Low-Code-Ansatz entwickelte Software bestmöglich zu testen, wäre für entsprechende Komponenten ein vom Anbieter bereitgestelltes, passendes Testframework ein optimaler Ansatz.

Im Fall von Microsoft Azure existiert eine Möglichkeit, Logic Apps zu testen, indem geprüft wird, ob der erwartete Responsecode empfangen wird. Da jedoch die korrekte Funktionalität der Logic App getestet werden soll, ist diese Art des Tests unzureichend, sodass im Folgenden eine eigene Möglichkeit des Testens der Logic App entwickelt wird.

## 5.2 Testfall: Slowly Changing Dimension 2

### 5.2.1 Ziel des Tests

Der Testfall, der im Rahmen dieser Arbeit implementiert wird, bezieht sich auf den ETL-Prozess. Dieser zieht sich über mehrere Komponenten des BI-Systems - von den Quellsystemen über den Data Lake, Logic Apps bis zu der Datenbank, in welcher die Daten als Ergebnis des Prozesses liegen. Wird der gesamte ETL-Prozess innerhalb eines Testes überprüft, wird dieser Testfall sehr komplex und widerspricht damit dem Ansatz, das Testen bei den kleinstmöglichen Einheiten zu beginnen. Die konkrete Funktionalität, die dabei getestet werden soll, ist die korrekte Änderung von Mitarbeiterdaten.

Wie in Kapitel 2.4.1 erläutert, existieren unterschiedliche Arten der Slowly Changing Dimensions. Im betreffenden Testsystem sind einige Attribute mit Typ 1 und weitere Attribute mit einer Mischform aus Typ 1 und Typ 2 umgesetzt. Im folgenden Beispiel werden der Standort sowie der Joblevel eines Mitarbeiters geändert.

Im Ausgangszustand der Tabelle (siehe Tabelle 5.1) befindet sich in dieser ein Datensatz, welcher die ID des Mitarbeiters, seinen Namen, Standort sowie Joblevel enthält. Ebenfalls enthalten sind die Daten des Gültigkeitszeitraumes. Da bisher noch keine Änderung an diesem Datensatz erfolgte, sind das frühestmögliche Anfangsdatum sowie das letztmögliche Enddatum angegeben. Der Datensatz ist als der aktuelle Datensatz markiert (1 - aktueller Datensatz; 0 - nicht mehr aktueller Datensatz).

Tabelle 5.1: Slowly Changing Dimension: Ausgangszustand

EmpID	Name	Standort	Joblevel	GueltigAb	GueltigBis	Aktueller Datensatz
emp_1	Julia Schneider	Jena	Trainee	01.01.1900	31.12.2999	1

Wird erstmalig in einem Monat eine Änderung an einem nach Typ 2 implementierten Attribut vorgenommen, erfolgt das Anlegen eines neuen Datensatzes. Dieser erhält als Datum des Beginns des Gültigkeitszeitraumes den ersten Tag des Monats der Änderung. In Tabelle 5.2 ist dargestellt, welche Datensätze enthalten sind, nachdem der Standort eines Mitarbeiters geändert wurde. Erfolgt die Änderung beispielsweise am 14.02.2022, ist das Ende des Gültigkeitszeitraumes des bisherigen Datensatzes der 31.01.2022, der Beginn des Gültigkeitszeitraumes des neuen Datensatzes ist der 01.02.2022. Der neue Datensatz ist als der aktuelle Datensatz markiert.

Tabelle 5.2: Slowly Changing Dimension: Historisierung eines Datensatzes nach Änderung des Standort-Attributes

EmpID	Name	Standort	Joblevel	GueltigAb	GueltigBis	Aktueller Datensatz
emp_1	Julia Schneider	Jena	Trainee	01.01.1900	31.01.2022	0
emp_1	Julia Schneider	Berlin	Trainee	01.02.2022	31.12.2999	1

Wird innerhalb des selben Monats eine weitere Änderung an diesem oder einem anderen Attribut vorgenommen, wird nur noch Typ 1 angewendet. In Tabelle 5.3 erfolgt die Änderung des Joblevel-Attributes. Erfolgt diese Änderung beispielsweise am 18.02.2022, wird kein weiterer Datensatz angelegt. Der Gültigkeitszeitraum bleibt auch in diesem Fall bei einem Anfangsdatum am 01.02.2022. Der Datensatz bleibt weiterhin als aktueller Datensatz markiert.

Tabelle 5.3: Slowly Changing Dimension: Historisierung eines Datensatzes nach Änderung des Joblevel-Attributes

EmpID	Name	Standort	Joblevel	GueltigAb	GueltigBis	Aktueller Datensatz
emp_1	Julia Schneider	Jena	Trainee	01.01.1900	31.01.2022	0
emp_1	Julia Schneider	Berlin	Junior	01.02.2022	31.12.2999	1

Dieser Test soll zeigen, dass die Historisierung bei einer Änderung des Loblevel-Attributes wie eben beschrieben funktioniert. Die korrekte Funktionsweise ist an folgenden Überprüfungen erkennbar:

- Der bisherige Datensatz ist nicht mehr als aktueller Datensatz markiert.
- Das Ende des Gültigkeitszeitraumes des bisherigen Datensatzes wird auf den letzten Tag des vorherigen Monats gesetzt.
- Es wird ein neuer Datensatz angelegt.
- Der neue Datensatz ist als aktueller Datensatz markiert.
- Der Anfang des Gültigkeitszeitraumes des neuen Datensatzes wird auf den ersten Tag des aktuellen Monats gesetzt.
- Der neue Datensatz enthält den aktualisierten Wert des Joblevels.
- Die Datei befindet sich nach der Durchführung des Tests nicht mehr in der Landing Zone des Data Lakes, sondern in der Archive Zone.

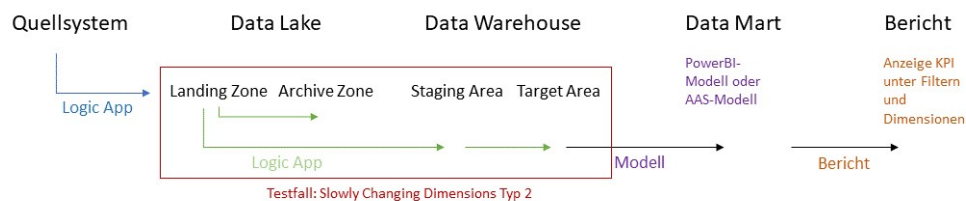


Abbildung 5.2: Einordnung des Testfalls in das Testsystem

Grafik 5.2 stellt dar, welche Abschnitte des Testsystems beansprucht werden, um diese Funktionalität durchzuführen. Die Schnittstelle zu dem Quellsystem wird in diesem Testfall außer Acht gelassen, somit wird auch die Logic App, welche der Übertragung vom Quellsystem in den Data Lake dient, nicht betrachtet. Die Betrachtung des Prozesses gilt ab dem Vorliegen der Daten als Datei in der Landing Zone des Data Lakes. Das Laden von dort bis zur Speicherung in der Target Area des Data Warehouses wird durch eine weitere Logic App geregelt. Diese steht im Fokus der Betrachtungen, da diese die gesamte erforderliche Logik enthält.

## 5.2.2 Umsetzung

Ein etablierter Ansatz des Testens in der klassischen Softwareentwicklung ist es, dass die einzelnen Bestandteile der Software von kleineren hin zu größeren getestet werden. Dabei werden immer die kleinstmöglichen, sinnvoll testbaren Bestandteile verwendet. Die Anzahl der Komponenten, die zusätzlich verwendet werden müssen, sollte so gering wie möglich gehalten werden. Sind komplexe Teile der Logik zu testen, erhöht sich die Komplexität der Tests entsprechend. Das Testen kleiner Bestandteile wirkt einer zu großen Komplexität entgegen. Funktionieren die einzelnen Bestandteile erwartungsgemäß, besteht eine hohe Wahrscheinlichkeit, dass diese wie erwartet zusammenarbeiten und das Gesamtsystem planmäßig funktioniert.

Da ein BI-System ebenso aus unterschiedlichen Bestandteilen besteht, die zusammenarbeiten, lässt sich davon ausgehen, dass dieses Prinzip auch auf das Testen eines BI-Systems anwendbar ist. Um den eben beschriebenen Anwendungsfall testen zu können, wird die Logic App benötigt,

welche alle benötigten Aktionen anstößt. Diese wird hier als zu testender Bestandteil betrachtet. Da die Logic App, wie bereits beschrieben, über keine eigene Testfunktionalität verfügt, muss eine andere Möglichkeit ermittelt werden, den Erfolg der Durchführung der Logic App zu prüfen. Die Ausführung der Logic App ist obligatorisch, um deren Funktion zu testen. Zur Überprüfung des Ergebnisses ist es nicht ausreichend, die erfolgreiche Durchführung der Logic App auf der Azure-Nutzeroberfläche zu kontrollieren. Dadurch ist nur sichergestellt, dass alle enthaltenen Aktionen ausgeführt wurden; nicht jedoch, ob sie auf die korrekte, erwartete Art ausgeführt wurden und das gewünschte Ergebnis bringen.

Die Logic App ist stark in das System integriert und arbeitet somit eng mit anderen Komponenten zusammen. In diesem Testfall ist das Ergebnis des Prozesses daran erkennbar, ob die Daten nach der erfolgreichen Ausführung der Logic App korrekt historisiert in der Mitarbeiter-Tabelle enthalten sind. Somit werden folgende weitere Komponenten benötigt, um die Funktionalität testen zu können:

- **Logic App:**  
Zum Starten der Logic App wird die Möglichkeit genutzt, diese über ihre URL aufzurufen, um sie zu starten. Da die erforderliche Logik für die Durchführung dieses Prozesses in der Logic App enthalten ist, ist es notwendig, die originale Logic App bzw. eine identische Version zu Testzwecken zu verwenden.
- **Datenbank:**  
Da getestet werden soll, ob durch die Änderung des Attributes im Quellsystem die erwarteten Änderungen in der Datenbank vorgenommen werden, ist es nötig, im Rahmen des Testes eine Datenbank zu verwenden. Dafür ist die Verwendung einer Datenbank mit einer Tabelle nötig, welche die zu ändernden Attribute sowie Attribute für die Markierung der Gültigkeit der einzelnen Datensätze enthält. In diesem Test wird eine Datenbank verwendet, die eine identische Kopie der originalen Datenbank darstellt.
- **Data Lake:**  
Der Test beginnt mit der Verarbeitung der Dateien aus dem Data Lake. Die Ordnerstruktur bildet die Grundlage für die Reihenfolge der Abarbeitung der Dateien, sollte somit also auch im Data Lake des Testsystems enthalten sein.

Um die Logic App zu testen, ergibt sich also die Notwendigkeit, weitere Komponenten in den Test einzubeziehen. Damit handelt es sich bei dem Test um keinen reinen Unit-Test mehr. Das Zusammenspiel aus mehreren Komponenten lässt auf einen Integrationstest schließen, bei welchen jedoch das Ziel ist, zu testen, ob die entsprechenden Komponenten untereinander harmonieren. In diesem Fall ist das Ziel hingegen, die Logic App zu testen, was, wie eben beschrieben, nur unter Zuhilfenahme anderer Komponenten möglich ist. Deshalb lässt sich dieser Test als eine Mischform aus Unit- und Integrationstest beschreiben.

Softwaretests können in unterschiedlichen Programmiersprachen geschrieben werden. Eine sehr beliebte Programmiersprache dafür ist Java. Um das Testen zu vereinfachen, wurden Frameworks entwickelt, welche unterstützende Funktionen beinhalten. Eines der beliebtesten Test-Frameworks, das beinahe schon als Standard-Framework gilt, ist *JUnit*. Unter der Verwendung von JUnit werden die Ergebnisse der einzelnen Tests übersichtlich aufgelistet. Dadurch lassen sich fehlgeschlagene Tests einfach erkennen und Fehler lassen sich schnell finden. Die Anwendung von JUnit ist simpel und somit auch für Anfänger geeignet.

Durch die Nutzung dieser Frameworks können mit geringem Aufwand Tests mit gut lesbarem

Code erstellt werden. Deshalb ist die Entscheidung für diesen Testfall auf einen Java-Test mit dem JUnit-Framework gefallen.

Der Test soll nun also die Logic App ausführen und prüfen, ob die erwarteten Daten in der Datenbank enthalten sind. Damit dieser Test durchgeführt werden kann, sind Schritte zur Vorbereitung dessen nötig:

1. Anlegen der Testdaten

Der Test soll mit realistischen Daten durchgeführt werden, weshalb es nötig ist, Testdaten nach dem Schema der Originaldateien im XML-Format zu erstellen. Diese beinhalten Werte für alle Pflichtattribute.

2. Einfügen der Testdaten in den Data Lake

Die Logic App bezieht die zu verarbeitenden Dateien aus der Landing Zone des Data Lakes. Bei der Reihenfolge der Verarbeitung wird das Datum des Ordners berücksichtigt, in welchem die Dateien liegen. Die zuerst zu verarbeitende Datei sollte sich demnach in einem Ordner befinden, der ein früheres Datum darstellt als der Ordner, in welchem sich die zweite Datei befindet.

Anschließend kann der Test ausgeführt werden, welcher die folgenden Schritte enthält:

1. Anlegen der Vergleichsdaten

Die Kontrolle, ob die Daten korrekt in die Datenbank eingefügt wurden, erfolgt durch den Vergleich dieser mit den erwarteten Daten. Dafür sind Vergleichsdaten nötig, die den Endzustand der Daten in der Datenbank darstellen. Das beinhaltet auch die Berücksichtigung der Slowly Changing Dimensions.

2. Herstellung der Datenbankverbindung

In diesem Test werden Aktionen aufgerufen, welche Datenbanktransaktionen ausführen. Die dafür nötige Verbindung zur Datenbank wird über die in Java integrierte Datenbankschnittstelle *JDBC* realisiert. Deren Hauptaufgaben bestehen in dem Aufbau und der Verwaltung von Datenbankverbindungen sowie dem weiterleiten und umwandeln von SQL-Anfragen, um sie in Java verwendbar zu machen. JDBC wird standardmäßig in der klassischen Softwareentwicklung verwendet und lässt sich ebenso beim Testen von BI-Systemen anwenden.

3. Datenbank in den gewünschten Startzustand bringen

Um bei jedem Test den gleichen Ausgangszustand vorliegen zu haben, werden zu Beginn jedes Testes alle Daten aus der betreffenden Tabelle entfernt.

4. Starten der Logic App

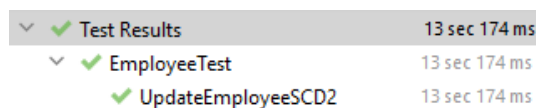
Nachdem die Vorbereitung der Datenbank abgeschlossen ist, kann die Logic App gestartet werden. Dies geschieht durch den Aufruf einer URL, welche der Logic App zugewiesen ist. Die Logic App lädt nun die erste Datei aus der Landing Zone des Data Lakes. Die Datensätze dieser Datei werden durch SQL-Prozeduren aus der Datei geladen, in die Tabelle der Staging Area und anschließend in die Tabelle der Target-Area des Data Warehouses geladen. Die zweite Datei wird aus dem Data Lake geladen. Die Datensätze dieser Datei werden, ebenso wie diejenigen aus der vorherigen Datei, bis in die Tabelle der Target-Area des Data Warehouses geladen. Dabei wird die Logik der Slowly Changing Dimensions berücksichtigt. Anschließend erfolgt das Verschieben der beiden Dateien von der Landing Zone des Data Lakes in die Archive Zone.

### 5. Abfrage der Daten aus der Datenbank

Nun ist zu prüfen, ob die Daten, die sich jetzt in der Datenbank befinden, den Daten entsprechen, die erwartet werden. Dafür werden alle Daten abgefragt, die sich in der Tabelle befinden. Die Daten werden in Objekten gespeichert, um sie im nächsten Schritt verwenden zu können.

### 6. Abgleich der erwarteten mit den tatsächlichen Daten

Aus den angelegten Vergleichsdaten und den aus der Datenbank geladenen Daten wird jedes Attribut über eine Assert-Anweisung verglichen. Erwartet wird dabei in jedem Fall eine vollständige Übereinstimmung der Werte.



✓ Test Results	13 sec 174 ms
✓ EmployeeTest	13 sec 174 ms
✓ UpdateEmployeeSCD2	13 sec 174 ms

Abbildung 5.3: Kennzeichnung des Testfalles als erfolgreich

Die Übereinstimmung aller Attribute führt zu einer Kennzeichnung der Tests als fehlerlos durchgeführt. (Abbildung 5.3) Das bedeutet, dass die Logik korrekt implementiert wurde und die Logic App wie erwartet arbeitet.

## 5.2.3 Auswertung

Der in diesem Kapitel beschriebene Test ist eine prototypische Umsetzung der Überlegungen, wie das Grundprinzip der klassischen Softwaretests speziell auf eine Logic App angewendet werden kann. Der Test ist wie in Kapitel 5.2.2 beschrieben implementiert und zeigt durch die Bewertung als *erfolgreich durchgeführt*, dass es möglich ist, einen Softwaretest in der Programmiersprache Java mit Hilfe des Frameworks JUnit auf eine Logic App anzuwenden. Daraus lässt sich schlussfolgern, dass sich diese Art von Tests auch auf andere Logic Apps in diesem BI-System anwenden lassen. Das Grundprinzip des Startens einer Logic App und des Vergleichens des tatsächlichen mit dem erwarteten Ergebnis lässt sich unabhängig vom Inhalt der Logic App anwenden.

Die Schwierigkeit beim Testen einer Logic App besteht in deren asynchroner Durchführung. Der Test wird ausgeführt und startet die Logic App. Während die Logic App ausgeführt wird, wird der Test weiter abgearbeitet. An der benötigten Stelle des Tests ist das Ende der Logic App jedoch noch nicht erreicht. Diese wird parallel weiterhin ausgeführt. Der Test würde an dieser Stelle fehlschlagen, da die durch die Logic App einzubringenden Daten noch nicht in der Datenbank vorhanden sind, sondern erst, wenn die Logic App abgeschlossen ist.

Für die Durchführung dieses Tests wurde nach dem Aufruf der Logic App eine *sleep*-Funktion eingebaut. Diese sorgt dafür, dass der Test erst weiter abgearbeitet wird, wenn eine bestimmte Anzahl Sekunden vergangen ist. Nach dieser Zeit sollte die Logic App vollständig durchgeführt worden sein. Diese Lösung ist jedoch nicht optimal, da diese Funktion einen starken Eingriff in das System darstellt und möglicherweise andere Aktivitäten, die auf dem selben System durchgeführt werden, beeinflussen kann.

Wird die Logic App im geschäftlichen Betrieb aufgerufen, geschieht das durch die Data Factory. Diese liefert beim Start eine Callback-URL mit, an welche die Logic App die Antwort nach



Fertigstellung sendet. Dadurch wird der Data Factory die Vollständige Durchführung bekannt gegeben.

Im Java Test wurde im Rahmen dieser Arbeit keine geeignete Möglichkeit gefunden, auf die Antwort der Logic App zu warten und erst im Anschluss den Test weiter abzuarbeiten. Auch die Möglichkeit, die Logic App synchron durchzuführen, besteht in diesem Fall nicht.

Die Callback-URL wird nicht nur verwendet, um das Ende der Durchführung der Logic App zu markieren. Sie ist zudem nötig, um die Logic App zu starten, da sie einen erwarteten Parameter in der HTTP-Anfrage an die URL dieser darstellt. Da die Callback-URL nicht aus der Data Factory generiert werden kann, da diese im Rahmen des Testes nicht benutzt wird, muss die Callback-URL auf eine andere Art generiert werden. Eine Möglichkeit stellt das Ermitteln eines Mocking-Frameworks dar, was die Erstellung einer Callback-URL ersetzen kann. Diese kann im POST-Request an die Logic App als Parameter verwendet werden.

Im Rahmen dieses Tests wurde eine Callback-URL verwendet, welche aus einem früheren Aufruf dieser Logic App durch die Data Factory stammt. Diese Lösung ist keine dauerhafte Lösung, ist jedoch ausreichend, um in dieser prototypischen Umsetzung sicherzustellen, dass der Test wie gewünscht funktioniert.

Ein möglicher Ansatz für die Lösung der eben genannten Schwierigkeiten ist die weitere Einschränkung des zu testenden Bereichs. Der hier verwendete Bereich, also die Logic App, lässt sich in weitere Bestandteile zerlegen. Durch das Testen kleinerer Bestandteile kann der Ansatz verfolgt werden, davon auszugehen, dass die Bestandteile auch in zusammengesetztem Zustand erwartungsgemäß funktionieren.

Einzelne Bestandteile, die sich in der Logic App befinden und getestet werden können, sind die Datenbankprozeduren.

Ein Test der Datenbankprozeduren würde, ebenso wie der Test der Logic App, das Anlegen von Test- und Vergleichsdaten sowie die Herstellung einer Datenbankverbindung benötigen. Der Start des Testes sollte ebenfalls bei einer leeren Datenbank erfolgen. Das Laden der Dateien und Starten der Datenbankprozeduren können direkt aus dem Test heraus erfolgen. Auch hier wird abschließend das Vergleichen der vorhandenen Daten mit den Vergleichsdaten durchgeführt.

## 6 Zusammenfassung und Ausblick

Die Erkenntnisse, die in dieser Arbeit zusammen getragen wurden, machen deutlich, dass es sich bei Business Intelligence-Systemen um Softwaresysteme handelt, die den gleichen Rahmenbedingungen unterliegen wie klassische Softwareprodukte. Dazu gehören beispielsweise sich ständig ändernde Datenstrukturen und sich ändernde Datenquellen. An Softwareprodukte wird die Anforderung gestellt, mit diesen Änderungen umgehen zu können. Da BI-Systeme den gleichen Rahmenbedingungen unterlegen sind, müssen auch diese auf Änderungen reagieren können. Der Unterschied dieser beiden Softwarearten ist geringer, als auf den ersten Blick erkennbar. Beide werden programmiert und von Nutzern bedient. Business Intelligence-Systeme sind jedoch keine klassisch verkauften Produkte, welche an mehrere Nutzer verkauft werden, sondern werden im Normalfall individuell für die jeweiligen Nutzer angefertigt.

Daraus ergibt sich, dass Business Intelligence-Systeme die gleichen Qualitätsstandards benötigen wie Softwareprodukte. Somit ist das Testen von Business Intelligence-Systemen unerlässlich. Die Notwendigkeit dafür ergibt sich aus den Problemen, zu denen ein unverlässliches BI-System führt, wie beispielsweise das Treffen von wichtigen Geschäftsentscheidungen auf der Basis von inkorrekten Zahlen und Fakten.

Die Untersuchungen haben ergeben, dass das Testen des BI-Systems erfolgen kann, indem Softwaretests angewendet werden, die ihren Ursprung in der OLTP-basierten Softwareentwicklung haben. Möglich sind dabei Tests aller Stufen, wobei die Umsetzung von Unit-Tests sich insofern schwierig gestaltet, als dass die Komponenten so eng miteinander verbunden sind, dass das Testen eines einzelnen Bestandteils nicht oder nur mit der Verwendung ausreichend vieler Mock-Funktionen umsetzbar ist. Möglich sind auch Mischformen der verschiedenen Teststufen. Zur Bestimmung der Qualität können die Metriken auf der klassischen Softwareentwicklung verwendet werden.

Eine Besonderheit stellt dabei die Testabdeckung des Codes des BI-Systems dar. Handelt es sich bei dem BI-System um ein vollständig selbst programmiertes System, lässt sich die Codeabdeckung einfach ermitteln; handelt es sich jedoch um ein cloudbasiertes System, bei welchem nur wenig Code für die Entwicklung des Systems nötig ist, lässt sich nicht mehr bestimmen, wie groß der Anteil an getestetem Code ist. Eine weitere Schwierigkeit dabei stellt die Verwendung unterschiedlicher Programmiersprachen und Technologien dar. Der Softwaretest kann in Java geschrieben sein; die getestete Logic App enthält keinen zugreifbaren Code und die darin verwendeten Datenbankprozeduren wurden in einer Datenbanksprache entwickelt. Dadurch ist die Messbarkeit der Testabdeckung stark beeinträchtigt.

Eine weitere Möglichkeit der Umsetzung des Tests wäre das Verfolgen eines Low- oder No-Code-Ansatzes. Das wäre durch die Verwendung von Tools möglich, welche sich speziell auf das Testen von BI-Systemen spezialisiert haben. Eine Auswahl an möglichen Tools ist in Kapitel 4.1 aufgeführt. Nachteilig daran wäre jedoch, dass die Verwendung solche Software mit Kosten verbunden ist. Werden die Tests selbst entwickelt und in Java geschrieben, können darauf ausgerichtete,

nicht-kommerzielle Frameworks verwendet werden, um die Tests kostenfrei umzusetzen. Ein weiterer Vorteil dieser Variante ist die simple Automatisierung dieser Tests. Bei vielen Anbietern der Versionsverwaltung ist eine CI/CD-Strecke integriert, mit welcher es möglich ist, in bestimmten Zeitintervallen bzw. nach Änderungen des Codes die Tests automatisch durchzuführen. Dadurch kann sichergestellt werden, dass das BI-System jederzeit korrekt funktioniert. Bei dieser Entscheidung muss abgewogen werden, welche Bestandteile das BI-System enthält, die sich testen lassen und wie das Verhältnis von Kosten und Nutzen steht.

Das konkrete Business Intelligence-System, welches in Kapitel 5 betrachtet wurde, ist mit Hilfe von Microsoft Azure umgesetzt. Dieses beinhaltet keine bereits integrierte Testfunktion, was die externe Entwicklung von Tests nötig macht. In Kapitel 5 wurde ein Test prototypisch umgesetzt. Dieser testet die Funktionalität einer im BI-System enthaltenen Logic App. Dieser Test eigent sich dazu, zu zeigen, dass es prinzipiell möglich ist, Softwaretests für das BI-System zu schreiben. Bei der Entwicklung dieses Tests haben sich jedoch Schwachstellen in Bezug auf die Testbarkeit ergeben.

Wie in Kapitel 5.2.3 beschrieben, besteht ein Problem darin, dass die Logic App asynchron abgearbeitet wird. Dies führt dazu, dass die nächsten Schritte des Tests bereits ausgeführt werden, bevor das dafür nötige Ergebnis der Logic App vorliegt. Es existiert keine Möglichkeit, die Logic App lokal auszuführen, da der Code nicht erreichbar ist. Im Rahmen dieser Arbeit konnte keine Möglichkeit ermittelt werden, wie dieses Problem umgangen werden kann. Ein Lösungsvorschlag an dieser Stelle ist die Auswahl eines kleineren Bestandteils, der getestet wird. Naheliegend ist in diesem Fall, in einem eigenen Test die in der Logic App enthaltenen Datenbankprozeduren zu testen. Aus diesem Sachverhalt lässt sich schließen, dass das Testen von orchestrierenden Aufgaben nicht optimal ist, solange keine Möglichkeit für die synchrone Ausführung dieser besteht. Das Testen der Teilaufgaben lässt sich jedoch nach dem im prototypischen Test gezeigten Prinzip gut umsetzen.

Eine weitere Möglichkeit für die Vereinfachung des Tests ist es, keine echte Datenbank zu verwenden. Es kann beispielsweise eine Excel-Tabelle benutzt werden. Dadurch wird die Anzahl der verwendeten Komponenten reduziert, was wieder eher dem Prinzip der Unit-Tests entspricht. Die Excel-Tabelle kann ebenfalls für eine weitere Art der Tests verwendet werden. Sind in dieser Beispieldaten enthalten, kann mit Hilfe der Analyse-Anwendung ein Modell erstellt werden, welches die Formeln zur Berechnung der Kennzahlen enthält. Durch dieses können testweise Kennzahlen mit vorgegebenen Daten berechnet werden. Die Ergebnisse der Berechnungen können anschließend mit den erwarteten Ergebnissen verglichen werden. Eine Übereinstimmung dieser Ergebnisse bedeutet die korrekte Implementierung der Formeln zur Berechnung sowie das korrekte Auslesen der Daten aus der Datenquelle.

Ein weiterer interessanter Ansatz für das Testen des BI-Systems ist die Verwendung von statistischen Tests. Wie in Kapitel 3 erläutert, können damit Abweichungen der hinzugefügten Daten von den durchschnittlichen Daten der bisher hinzugefügten Daten ermittelt werden. Diese Möglichkeit findet in der klassischen Softwareentwicklung keine große Anwendung, scheint durch den besonderen Bezug auf die Daten jedoch sehr gut für BI-Systeme geeignet zu sein.

Die abschließende Empfehlung in dieser Arbeit lautet, die Testbarkeit des Systems bereits bei der Planung zu berücksichtigen. Im Falle des in Kapitel 5 untersuchten BI-Systems wurde die Testbarkeit nicht ausreichend berücksichtigt. Es wurden verschiedene Technologien für die Umsetzung des Data Warehouses verwendet, ohne die Testbarkeit zu beachten. Da keine Testfunktion integriert ist, muss diese nun durch die Entwicklung externer Tests ersetzt werden.

Dieses Vorhaben ist durch die Verwendung der unterschiedlichen Technologieebenen erschwert. Wären weniger technologische Brüche im BI-System enthalten, würde die Möglichkeit der Verwendung eines einzigen Testtools für die verschiedenen Ebenen bestehen.

Wird also ein Cloud-System verwendet, gestaltet sich die Entwicklung der Tests schwieriger als würde direkter Zugriff auf alle Teile des Codes bestehen. Trotz dieser Schwierigkeiten lässt sich die eingangs gestellte Frage, ob Business Intelligence-Systeme mit den Methoden der OLTP-basierten Softwareentwicklung getestet werden können, mit ja beantworten.

## Literaturverzeichnis

- [AFF90] ALLEN, R.E. ; FOWLER, H.W. ; FOWLER, F.G.: *The Concise Oxford Dictionary of Current English*. Clarendon Press, 1990 (Oxford English dictionary word and language service). <https://books.google.de/books?id=Q8ThbE08zTEC>. – ISBN 9780198612001
- [AG] AG, SYNAXON: *Slowly Changing Dimensions (SCD) im Data Warehouse*. <https://synaxon.ag/blog/2012/11/02/slowly-changing-dimensions-scd-im-data-warehouse/>
- [Ale] ALEXANDERTHAMMGMBH: *Grundlagen, Anwendungsfälle und Vorzüge eines Data Lake: Alles was Unternehmen über Data Lakes wissen müssen*. <https://www.alexanderthamm.com/de/blog/grundlagen-anwendungsfaelle-data-lake/>, Abruf: 06. 12. 2021.
- [Alk14] ALKHATEEB, Jawad: *cost quality*. [https://www.researchgate.net/publication/260391390\\_cost\\_quality](https://www.researchgate.net/publication/260391390_cost_quality). Version: 02 2014
- [AM97] ANAHORY, Sam ; MURRAY, Dennis: *Data Warehouse: Planung, Implementierung und Administration*. Bonn : Addison-Wesley Verlag, 1997. – ISBN 978-3-827-31288-4
- [Ama] AMAZON: *Beginnen Sie die Erstellung mit AWS*. [https://aws.amazon.com/de/?nc2=h\\_lg](https://aws.amazon.com/de/?nc2=h_lg)
- [AQ] AL-QUTAISH, Rafa E.: *An Investigation of the Weaknesses of the ISO 9126 International Standard*. <https://www.semanticscholar.org/paper/An-Investigation-of-the-Weaknesses-of-the-ISO-9126-Al-Qutaish/bc5114622f3a7057a188b9b8fa081763b64332d7>, Abruf: 12. 01. 2021.
- [Are18] AREZINA, Nebojsa: *Data Lake - the evolution of data processing*. <https://www.kdnuggets.com/2018/06/data-lake-evolution-data-processing.html>. Version: 06 2018
- [ASD14] ALVES, Tiago L. ; SILVA, Pedro ; DIAS, José Miguel S.: Applying ISO/IEC 25010 Standard to Prioritize and Solve Quality Issues of Automatic ETL Processes. In: *2014 IEEE International Conference on Software Maintenance and Evolution (2014)*, S. 573–576
- [BBL76] BOEHM, Barry W. ; BROWN, John R. ; LIPOW, Mlity: Quantitative evaluation of software quality. In: *Proceedings of the 2nd international conference on Software engineering*, 1976, S. 592–605

- [BG13] BAUER, Andreas ; GUENZEL, Holger: *Data-Warehouse-Systeme: Architektur, Entwicklung, Anwendung*. 4. Heidelberg : Dpunkt-Verlag, 2013. – ISBN 978–3–898–64785–4
- [Dat] DATAWAREHOUSE4U: *What are Slowly Changing Dimensions?* <https://www.datawarehouse4u.info/SCD-Slowly-Changing-Dimensions.html>
- [Den91] DENERT, Ernst: *Software-Engineering: Methodische Projektentwicklung*. Berlin, Heidelberg : Springer Berlin Heidelberg, 1991. – ISBN 978–3–642–84343–3
- [Dev97] DEVLIN, Barry: *Data Warehouse: From Architecture to Implementation*. Addison-Wesley, 1997. – ISBN 978–0201964257
- [Dev18] DEVLIN, Barry: Thirty Years of Data Warehousing. In: *Business Intelligence Journal* 23 (2018), S. 12–24
- [DF95] DECKER, Karsten M. ; FOCARDI, Sergio: Technology Overview: A Report on Data Mining / Swiss Scientific Computing Center. Version: May 1995. <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=79A8EC5D4DB9BC3C43592E705B2F971D?doi=10.1.1.50.6800&rep=rep1&type=pdf>. CH-6928 Manno : Swiss Scientific Computing Center, May 1995 (CSCS TR-95-02). – Forschungsbericht. – 31 S.
- [DGH03] DUSTDAR, Schahram ; GALL, Harald ; HAUSWIRTH, Manfred: *Software-Architekturen für Verteilte Systeme*. Springer, 2003. – ISBN 978–3–642–62769–9
- [Dix10] DIXON, James: *Pentaho, Hadoop, and Data Lakes*. <https://jamesdixon.wordpress.com/2010/10/>. Version: 10 2010
- [eBu15] EBUSINESSLOTSE: *Business-Intelligene-Architektur. Eine Basis zur erfolgreichen Datensammlung und -auswertung*. Booklet, 2015
- [Gar84] GARVIN, David: What does product quality really mean? In: *Sloan Management Review* 26 (1984), 25–45. <http://doku.iab.de/externe/2006/k060210f02.pdf>
- [GGH<sup>+</sup>20] GIEBLER, Corinna ; GRÖGER, Christoph ; HOOS, Eva ; EICHLER, Rebecca ; SCHWARZ, Holger ; MITSCHANG, Bernhard: Data Lakes auf den Grund gegangen. In: *Datenbank Spektrum* 20 (2020), 01, S. 57–59
- [Gil11] GILLIES, Alan: *Software Quality: Theory and Management*. Lulu.com, 2011 <https://books.google.de/books?id=XTvpAQAAQBAJ>. – ISBN 978–1–446–75398–9
- [Goo] GOOGLE: *BigQuery*. <https://cloud.google.com/bigquery>
- [GR11] GOLFARELLI, Matteo ; RIZZI, Stefano: Data Warehouse Testing. In: *IJDWM* 7 (2011), 04, S. 26–43. <http://dx.doi.org/10.4018/jdwm.2011040102>. – DOI 10.4018/jdwm.2011040102
- [Hah14] HAHNE, Michael: *Modellierung von Business-Intelligence-Systemen. Leitfaden für erfolgreiche Projekte auf Basis flexibler Data-Warehouse-Architekturen*. 1. Heidelberg : dpunkt.verlag, 2014. – ISBN 978–3–89864–827–1

- [Har] HARASYMCZUK, Matt: *Quality*. <https://dev.astrotech.io/sonarqube/quality-models.html#jim-mccall-software-quality-model-1977>, Abruf: 14. 02. 2022.
- [Hof13] HOFFMANN, Dirk W.: *Software-Qualität*. Springer-Verlag, 2013
- [Hö] HÖCHT, Susanne: *N-tier-Softwarearchitektur*. <https://wiki.tum.de/display/logistikkompendium/N-tier-Softwarearchitektur>, Abruf: 07. 12. 2021.
- [IS] IT-SERVICE.NETWORK: *Was ist Software?* <https://it-service.network/it-lexikon/software>, Abruf: 02. 12. 2021.
- [ISO01] ISO/IEC: *ISO/IEC 9126. Software engineering – Product quality*. Geneva, Switzerland : ISO/IEC, 2001
- [ISO08a] ISO/IEC: *ISO/IEC 25012. Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Data quality model*. Geneva, Switzerland : ISO/IEC, 2008
- [ISO08b] ISO/IEC: *ISO/IEC 9000. Quality management systems — Fundamentals and vocabulary*. Geneva, Switzerland : ISO/IEC, 2008
- [ISO08c] ISO/IEC: *ISO/IEC 9001. Quality management systems — Requirements*. Geneva, Switzerland : ISO/IEC, 2008
- [ISO11] ISO/IEC: *ISO/IEC 25010. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. Geneva, Switzerland : ISO/IEC, 2011
- [KBM10] KEMPER, Hans-Georg ; BAARS, Henning ; MEHANNA, Walid: *Business Intelligence - Grundlagen und praktische Anwendungen*. 3. Wiesbaden : Vieweg + Teubner, 2010. – ISBN 978–3–8348–0719–9
- [KMU07] KEMPER, Hans-Georg ; MEHANNA, Walid ; UNGER, Carsten: *Business Intelligence - Grundlagen und praktische Anwendungen. Eine Einführung in die IT-basierte Managementunterstützung*. 2. Wiesbaden : Vieweg + Teubner, 2007. – ISBN 978–3–8348–9056–6
- [KSS14] KÖPPEN, Veit ; SAAKE, Gunter ; SATTLER, Kai-Uwe: *Data Warehouse Technologien*. 2. Ilmenau : mitp, 2014. – ISBN 978–3–8266–9485–1
- [LG02] LEROUGE, Cindy ; GJESTLAND, Creggan: *A Typology Of Data Warehouse Quality / University of South Florida; University of Alabama*. 2002. – Forschungsbericht
- [Luba] LUBER, Stefan: *Was ist ein Data Lake?* <https://www.bigdata-insider.de/was-ist-ein-data-lake-a-686778/>, Abruf: 06. 12. 2021.
- [Lubb] LUBER, Stefan: *Was ist ein Data Warehouse?* <https://www.bigdata-insider.de/was-ist-ein-data-warehouse-a-606701/>, Abruf: 07. 12. 2021.
- [Mic] MICROSOFT: *Lokal, Hybrid Cloud, Multi Cloud oder Edge: zukunftsweisende Cloud-lösungen in Azure entwickeln*. <https://azure.microsoft.com/de-de/>

- [MR14] MEULEN, Rob Van d. ; RIVERA, Jessica: *Gartner Says Beware of the Data Lake Fallacy*. [gartner.com/en/newsroom/press-releases/2014-07-28-gartner-says-beware-of-the-data-lake-fallacy](https://gartner.com/en/newsroom/press-releases/2014-07-28-gartner-says-beware-of-the-data-lake-fallacy). Version: 07 2014
- [MRW77] MCCALL, Jim A. ; RICHARDS, Paul K. ; WALTERS, Gene F.: *Factors in software quality. volume i. concepts and definitions of software quality / GENERAL ELECTRIC CO SUNNYVALE CA. 1977. – Forschungsbericht*
- [mysa] MYSERVERNAME: *Tutorial zum Testen von ETL-Data Warehouse-Tests (Eine vollständige Anleitung)*. [https://de.myservername.com/etl-testing-data-warehouse-testing-tutorial#Difference\\_between\\_Database\\_and\\_Data\\_Warehouse\\_Testing](https://de.myservername.com/etl-testing-data-warehouse-testing-tutorial#Difference_between_Database_and_Data_Warehouse_Testing)
- [mysb] MYSERVERNAME: *Vollständige Anleitung zum Testen von Datenbanken (Warum, Was und Wie Daten testen)*. <https://de.myservername.com/database-testing-complete-guide-why>
- [Onp] ONPULSON: *Human Engineering*. <https://www.onpulson.de/lexikon/human-engineering/>, Abruf: 12. 01. 2021.
- [Per] PERSONIOGMBH: *Personalcontrolling: Ziele, Instrumente, Kennzahlen*. <https://www.personio.de/hr-lexikon/personalcontrolling/#1>, Abruf: 08. 12. 2021.
- [pmq] PMQS: *Grundlagen der Qualitätssicherung*. <https://pmqs.de/index.php/qualitaetssicherung/grundlagen-qs/21-grundlagen-der-qualitaetssicherung>
- [Que] QUERYSURGE: *Data Warehouse Testing*. <https://www.querysurge.com/>
- [Rig] RIGHTDATA: *ETL Testing*. <https://getrightdata.com/index.php>
- [Sar] SARKAR, Sumit: *Wie unterscheiden sich Data Lakes von Data Warehouses?* <https://www.bigdata-insider.de/wie-unterscheiden-sich-data-lakes-von-data-warehouses-a-562102/>, Abruf: 07. 12. 2021.
- [SBGB12] SEIDL, R. ; BUCSICS, T. ; GWIHS, S. ; BAUMGARTNER, M.: *Basiswissen Testautomatisierung: Konzepte, Methoden und Techniken*. dpunkt.verlag, 2012. – ISBN 9783864917066
- [Sch13] SCHMITZ, Paul: *Software-Qualitätssicherung - Testen im Software-Lebenszyklus*. 2. Braunschweig : Springer-Verlag, 2013. – ISBN 978-3-3228-6225-9
- [Sch20] SCHMERLER, Stefan: *Softwaretest in der Praxis: Grundlagen, Methoden und Technologien*. Berlin : epubli, 2020
- [Sof] SOFTGUIDEGMBHU.CO.KG: *Business Intelligence-Definition*. <https://www.softguide.de/software-tips/business-intelligence-definition>, Abruf: 04. 12. 2021.
- [Ste11] STENDER, Peter: *Kurze Einführung in die OO-Programmierung*. Wiesbaden : Vieweg+Teubner Verlag, 2011. – ISBN 978-3-8348-8114-4



- [Tala] TALEND: *Bereinigte, vollständige und korrekte Daten für alle*. <https://www.talend.com/de/>
- [Talb] TALEND: *ETL Testing: An Overview*. <https://www.talend.de/resources/etl-testing/>
- [Talc] TALENDGERMANYGMBH: *Extract Load Transform (ELT) – Definition, Ablauf und Vorteile*. <https://www.talend.com/de/resources/was-ist-elt/>, Abruf: 07.12.2021.
- [TZ16] TRAHASCH, Stephan ; ZIMMER, Michael: *Agile Business Intelligence - Theorie und Praxis*. Heidelberg : dpunkt.verlag, 2016. – ISBN 978–3–864–91873–5