MASTERARBEIT

ITERATIVES TESTEN VON BI-SYSTEMEN

Entwicklung einer Teststrategie

ausgeführt an der

am Studiengang

Software Engineering Leadership

Von: Jonas Assies

Personenkennzeichen: 1540030001

Münster, 03.08.2018 .....

Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

ABSTRACT

Die vorliegende Masterthesis liefert eine Einführung in die Themen Agile Business Intelligence, Data-Warehouse Architektur und das Testen von Software. Darauf aufbauend werden grundlegende Konzepte für das Testen von Data-Warehouse Systemen analysiert, und Ansätze für die Automatisierung solcher Systeme herausgearbeitet. Inhaltlich wird über die Thematiken Agilität und iterative Entwicklungsmethoden an die Notwendigkeit eines Konzeptes für die Testautomatisierung im Data-Warehouse Umfeld herangeführt.

GLEICHHEITSGRUNDSATZ

Aus Gründen der Lesbarkeit wurde in dieser Arbeit darauf verzichtet, geschlechtsspezifische Formulierungen zu verwenden. Jedoch möchte ich ausdrücklich festhalten, dass die bei Personen verwendeten maskulinen Formen für beide Geschlechter zu verstehen sind.

INHALTSVERZEICHNIS

1 EINLEITUNG

1.1 Motivation und Ziel der Arbeit

Business Intelligence (BI) als unternehmensspezifischer Ansatz zur Entscheidungsunterstützung ist Kernbestandteil der IT-Infrastruktur von Unternehmen. Als Voraussetzung für eine faktenbasierte Entscheidungsfindung hat BI erheblichen Einfluss auf eine erfolgreiche Unternehmenssteuerung (Trahasch & Zimmer, 2016, S. 1). Eine hohe Dynamik in unterschiedlichen Wirtschaftszweigen erfordert, dass Entscheidungen häufiger und schneller zu treffen sind. Diese Marktsituation setzt Flexibilität und Adaptionsfähigkeit der dispositiven Systeme voraus (Krawatzeck, Zimmer, & Trahasch, 2013, S. 56). Um der permanenten Veränderung zu begegnen, haben sich im Bereich des Softwareengineerings agile Werte und Prinzipien etabliert. Als agil werden leichtgewichtige Vorgehensweisen zur Softwareentwicklung bezeichnet. Die Werte des Agile Manifests (Beck, et al., 2001) bilden das Fundament der agilen Software Entwicklung:

- Individuen und Interaktionen mehr als Prozesse und Werkzeuge
- Funktionierende Software mehr als umfassende Dokumentation
- Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
- Reagieren auf Veränderung mehr als das Befolgen eines Plans

Aus diesen agilen Werten wurden agile Prozesse (Vorgehensmodelle) wie Scrum (Schwaber & Sutherland, 2013) und agile Methoden wie Continous Integration (Duvall, Matyas, & Glover, 2013) abgeleitet. Jedoch sind die agilen Methoden aufgrund der Unterschiede zwischen klassischen Anwendungsentwicklungsprojekten und dispositiven Systeme nicht direkt auf diese übertragbar (Krawatzeck, Zimmer, & Trahasch, 2013, S. 57). Unterschiede sind dabei zum Beispiel eine erschwerte Anforderungsanalyse, abteilungsabhängige fachliche Begriffsdefinitionen sowie eine hohe Vernetzung der abteilungsübergreifenden beteiligten Systeme (Stauffer, Honegger, & Gisin, 2013, S. 8f.). Die Adaption agiler Werte und Prinzipien auf den Bereich der dispositiven Systeme wird unter den Themenfeldern agile BI (Trahasch & Zimmer, 2016) und agile Analytics (Collier, 2012) behandelt. Dabei ist agile BI nicht allein auf die Auswahl und Anwendung eines agilen Vorgehensmodells zur Produktentwicklung beschränkt, sondern schließt zudem Maßnahmen aus den Bereichen Prinzipien, Methoden und Technologie ein. Als eine technische Maßnahme zur Qualitätsprüfung und zur Gewährleistung der Korrektheit von vorab vorhandenen und neuen Funktionalitäten, gilt die automatisierte Testdurchführung (Krawatzeck, Zimmer, & Trahasch, 2013, S. 59). Unter Automatisierung von Tests ist die Durchführung von ansonsten manuellen

Testtätigkeiten zu verstehen (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 7). „Gerade BI-Lösungen setzten sich aus mehreren, sehr umfangreichen Systemteilen zusammen. Eine kontinuierliche Überprüfung der Korrektheit dieser Systemteile nach Änderungen ist innerhalb kurzer Entwicklungszyklen manuell kaum möglich." (Trahasch & Zimmer, 2016, S. 112)

Als Berater und Entwickler für BI-Systeme muss ich häufig bereits bestehende Anwendungen weiterentwickeln. In den seltensten Fällen bekomme ich die Möglichkeit ein System von Grund auf neu zu entwickeln. In den Projekten, in denen ich tätig war, ist es bisher nicht vorgekommen, dass für bestehende Systemteile, automatisierte Testfälle vorlagen. Im besten Fall lag eine Dokumentation über in der Vergangenheit durchgeführte Tests vor. Befragungen meiner Kollegen und ein Blick in die Literatur bestätigten meinen ersten Eindruck. Trotz der Relevanz des Testens von BI-Systemen, wird es in BI-Fachbüchern kaum adressiert (Stauffer, Honegger, & Gisin, 2013, S. 9). Der Eindruck wird zudem durch eine im Jahr 2013 durchgeführte Umfrage unter deutschsprachigen BI-Professionals untermauert. Lediglich knapp mehr als die Hälfte der befragten Unternehmen (53 %; n=55) testen ihr BI-System in irgendeiner Art und Weise (Krawatzeck, Zimmer, & Trahasch, 2013, S. 11).

Die zentrale Frage dieser Arbeit ist deshalb, wie mit Hilfe von automatisierten Tests sichergestellt werden kann, dass bei einer iterativen Erweiterung von BI-Systemen, bestehende Komponenten ohne manuellen Retest weiterhin funktionieren. Die untergeordneten Fragen, welche Testpunkte und Testobjekte relevant für das Testen sind, welche konzeptionellen Ansätze es bereits für die Automatisierung, speziell im BI-Umfeld gibt, und ob sich Methoden aus der klassischen Softwareentwicklung adaptieren lassen, sollen zur Beantwortung der Fragestellung führen. Ziel ist es, mithilfe eines argumentativ deduktiven Vorgehens herauszufinden, wie sich ein BI-System so testen lässt, dass bei einer Erweiterung sichergestellt ist, dass der bestehende Teil des Systems weiterhin funktioniert. Das Ergebnis ist eine Teststrategie die eine agile Entwicklung unterstützt.

1.2 Aufbau der Arbeit

Die Arbeit liefert eine Einführung in die Themen agile BI, Data-Warehouse-Architektur und Softwaretesten. Darauf aufbauend werden grundlegende Konzepte für das Testen von BI-Systemen analysiert, und Ansätze für die Automatisierung solcher Systeme herausgearbeitet. Die Notwendigkeit von automatisierten Tests wird besonders in sich schnell verändernden Umgebungen deutlich. Deshalb beginnt diese Arbeit mit einer Vorstellung von BI-Agilität als Eigenschaft der BI einer Organisation, „vorhersehbare und unvorhersehbare Anforderungen in Bezug auf Funktionalität oder den Inhalt einer BI-Lösung in einem vorgegebenen Zeitrahmen in angemessener Qualität abzubilden" (Krawatzeck, Zimmer, & Trahasch, 2013, S. 3), und agiler BI als Maßnahmen eines Unternehmens, die durchgeführt werden, um die BI-Agilität zu erhöhen. Die agilen Werte

und Prinzipien als Fundament für BI-Agilität und agile BI stehen dabei im Fokus. Aus den Werten und Prinzipien lassen sich Maßnahmen und Methoden für die Entwicklung von BI-Systemen ableiten. Dies sind organisatorische Maßnahmen, wie eine enge Verzahnung der Fach- und IT-Abteilung sowie eine im Unternehmen verankerte Feedbackkultur. Aber auch technische Maßnahmen, zu denen eine flexible BI-Architektur zählt, sind von Bedeutung. Die Vorstellung und der Vergleich verschiedener agiler Entwicklungsmethoden wie Scrum oder XP, würden den Umfang dieser Arbeit zu sehr strapazieren. Zumal Testautomatisierung nicht an eine Methode gebunden ist. Eine Einführung in Scrum findet sich in (Schwaber & Sutherland, 2013). Adaptionen von Scrum auf den Bereich Business Intelligence werden unter anderem von (Collier, 2012) und (Hughes, 2016) vorgenommen. Im deutschsprachigen Raum bieten (Trahasch & Zimmer, 2016) einen guten Einblick. BI-Systeme sind auf mehrschichtigen Architekturen aufgebaut. Um eine Vorstellung der Komplexität von BI-Testen zu vermitteln, werden die einzelnen Schichten und ihre Aufgaben vorgestellt. Zudem wird auf die zu berücksichtigenden Qualitätsmerkmale eingegangen (Hahne, 2014). Eine Diskussion von verschiedenen Architekturvarianten und deren Vorzügen hat bereits hinreichend an anderer Stelle stattgefunden und ist deshalb nicht Bestandteil dieser Arbeit. Ausgehend vom allgemeinen Prozess des Softwaretestens wird in das Testen von Software eingeführt. Der fundamentale Testprozess besteht aus fünf einzelnen Testphasen: (1) Testplanung und Steuerung, (2) Testanalyse und Testdesign, (3) Testrealisierung und Testdurchführung, (4) Testauswertung und Bericht sowie (5) Abschluss der Testaktivitäten (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 18 ff.). Der Testprozess ermöglich ein strukturiertes Testvorgehen. In Anlehnung an die Entwicklungsschritte im V-Modell (Alpar, et al., 2014, S. 329) werden beim Softwaretest die vier Teststufen Komponententest, Integrationstest, Systemtest und Abnahmetest unterschieden (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 41). Die genannten Testbegrifflichkeiten, sofern nicht weiter definiert, entsprechen der Definition des (ISTQB, 2017). BI Agilität, die BI-Architektur und Softwaretesten bilden die Einflussdimensionen für das automatisierte BI-Testen. Kapitel 5stellt verschiedene Ansätze für das Testen von BI-Systemen vor. Hierbei wird herausgearbeitet, mit welchen methodischen Ansätzen die Autoren auf die besonderen Herausforderungen, insbesondere der heterogenen Systemlandschaft und der Datenzentriertheit, umgehen. Zunächst wird gezeigt, welche logischen Testfälle sich für die Autoren aus der Architektur, den Testzielen, und Teststufen ergeben. Hierzu werden die einzelnen Ansätze vorgestellt und anschließend in den wichtigsten Punkten gegenübergestellt. Ergebnis dieser Arbeit ist eine Teststrategie die als Basis für projektspezifische Testkonzepte dienen kann.

1.3 Ähnliche Forschungsvorhaben und Abgrenzung

Stauffer, Honegger und Gisin liefern eine umfassende Einführung in das Thema BI-Testen. Dabei wird der Umfang der Testfälle anhand der drei Einflussdimensionen (1) Teststufe, (2) Architekturstufe und Qualitätsmerkmal hergeleitet. (Stauffer, Honegger, & Gisin) orientieren sich dabei an den vom (ISTQB) vorgeschlagenen Prinzipien (Stauffer, Honegger, & Gisin, 2013). Trahasch & Zimmer greifen diesen Ansatz auf und bilden aus diesen Dimensionen einen „*BI-Testing-Cube*“. Anhand dieses Würfels werden die Möglichkeiten der Testautomatisierung veranschaulicht. Dabei stellen Trahasch & Zimmer das Thema BI-Testautomatisierung in den Kontext Agile BI (Trahasch & Zimmer, 2016). In diesen Kontext stellen auch Hughes (Hughes, 2016) und Collier (Collier, 2012) das Thema BI-Testautomatisierung. Sie leiten die Teststufen und Automatisierungsgrade anhand einer 2x2 Matrix her. Die vier Dimensionen (1) Geschäftsakzeptanz *(Business acceptability*), (2) Produktvalidität *(Product validation), (3)* technische Akzeptanz *(Technical acceptability)* und (4) Systemvalidierung *(System validation),* sollen sicherstellen, dass das richtige Produkt auf die technisch richtige Weise umgesetzt wird (Collier, 2012). Das Ergebnis dieser Arbeit soll als Unterstützung für die konkrete Testplanung in BI Projekten dienen. Dabei wird zum einen darauf eingegangen wie Testfälle auf die einzelnen Integrationsstufen und die Integrationsstufen innerhalb der Iteration zu verteilen sind. Ein weiterer Aspekt sind methodische Ansätze, um Testfälle anhand von Risiken und Qualitätsmerkmalen zu ermitteln. Kein Ziel dieser Arbeit ist hingegen die technische Konzeption des BI-Testens. Beispiele für den technischen Aufbau einer Testautomatisierung sind bei (Bucsics, Baumgartner, Seidl, & Gwihs, 2015) und bei (Hughes, 2016) zu finden. Konzepte wie Test-Driven-Development (TDD), oder Continuous Integration (CI) (Duvall, Matyas, & Glover, 2013) werden im Rahmen dieser Arbeit nicht betrachtet. Einen Ansatz TDD auf den Kontext BI zu übertragen, wird bei (Sam Schutte, Thilini Ariyachandra, & Mark Frolick, 2013) formuliert. Sie sehen Testautomatisierung als notwendige Voraussetzung für TDD, beschreiben jedoch keine Methode für die Implementierung einer Automatisierung.

2 AGILE BUSINESS INTELLIGENCE

2.1 BI-Agilität und Agile Business Intelligence

Unter BI subsumierte Gesamtansätze zur Entscheidungsunterstützung stehen vor der Aufgabe sowohl übergreifende Anforderungen nach Standardisierung als auch bereichsspezifischer Agilitätsanforderungen abzubilden (Trahasch & Zimmer, 2016, S. 3). Unter BI Agilität wird die Eigenschaft der BI einer Organisation verstanden, „vorhersehbare und unvorhersehbare Anforderungen in Bezug auf Funktionalität oder den Inhalt einer BI-Lösung in einem vorgegebenen Zeitrahmen in angemessener Qualität abzubilden." (Krawatzeck, Zimmer, & Trahasch, 2013, S. 3) Dabei sind sowohl die BI-Architektur, die BI-Aufbauorganisation als auch die zugehörigen BI-Prozesse zu betrachten. Alle Maßnahmen eines Unternehmens, die durchgeführt werden, um die BI-Agilität zu erhöhen, werden unter dem Begriff Agile Business Intelligence (Agile BI) zusammengefasst (Krawatzeck, Zimmer, & Trahasch, 2013, S. 3). Agile BI beschränkt sich dabei nicht nur auf die Auswahl und Anwendung eines Agilen Vorgehensmodells zur Produktentwicklung, sondern es schließt Maßnahmen aus den Bereichen Prinzipien, Methoden, und Technologie ein. Die Eigenschaft der BI und Agile BI als konkrete Maßnahmen werden in Abb. 2-1 dargestellt.

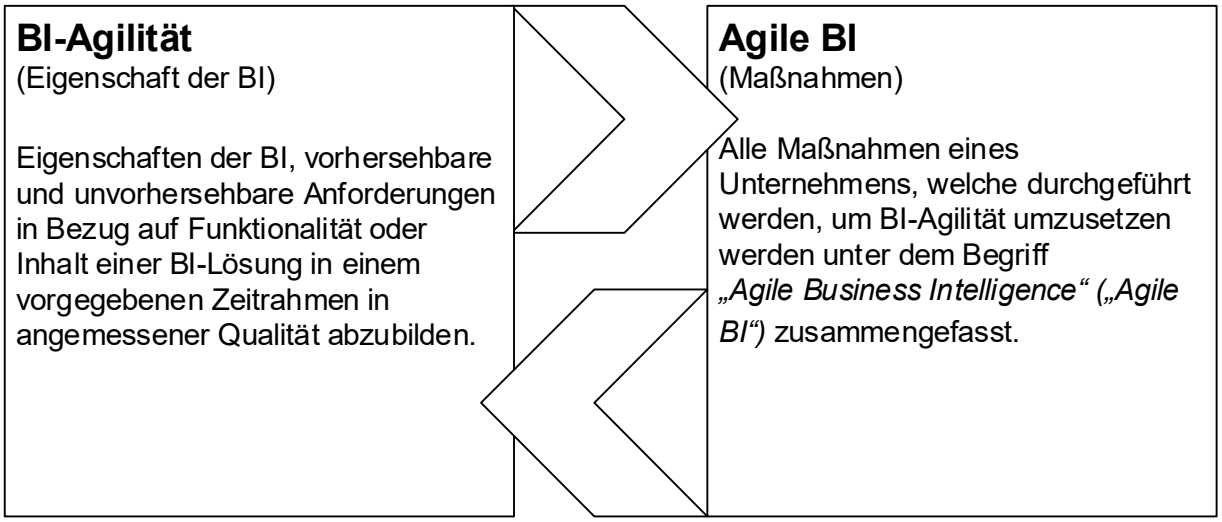

Abb. 2-1 BI-Agilität als Eigenschaft der BI und Agile BI als konkrete Maßnahmen (Trahasch & Zimmer, 2016)

Die Notwendigkeit einer hohen BI-Agilität muss sich aus der Unternehmensstrategie ableiten und durch die Anforderungen des Fachbereichs begründen lassen. Variationsmöglichkeiten von Agile-BI-Maßnahmen werden unmittelbar durch die BI-Architektur, die Aufbauorganisation, die BI-Prozesse und letztlich die BI-Governance reglementiert (Zimmer, 2015). Innerhalb dieses Gestaltungsspielraums können Maßnahmen in den Bereichen Prinzipien, Vorgehensmodelle, Methoden und Technologien festgelegt und umgesetzt werden. Prinzipien legen Grundsätze fest,

nach denen gehandelt werden soll. Methoden und Vorgehensmodelle liefern definierte Rahmenbedingungen, Rollenverantwortlichkeiten Abläufe, Abschnitte und Artefakte (Trahasch & Zimmer, 2016, S. 5).

2.2 Werte und Prinzipen

Agilität setzt eine offene und konstruktive Kommunikation, eine hohe Selbstverantwortung des Teams und der Stakeholder sowie ständige Lernbereitschaft voraus. Grundlagen für diese Haltung werden im Manifest für Agile Softwareentwicklung definiert (Beck, et al., 2001). Die Entwicklung von BI Anwendungen unterscheidet sich von der Softwareentwicklung. Deswegen wurden die bekannten Werte und Prinzipien für BI angepasst. Diese stellen den Bezugsrahmen, auf dessen Basis Agile BI Maßnahmen umgesetzt werden können (Trahasch & Zimmer, 2016, S. 6).

2.2.1 Werte

Die Agilen BI Werte sind das Fundament für die erfolgreiche Umsetzung von Agile BI. Alle Entscheidungen und Handlungen in Agilen BI-Projekten basieren auf diesem Fundament. Dies bedeutet, dass für alle Beteiligten die Verinnerlichung des Wertesystems notwendig ist. Daraus folgt eine Veränderung der gewohnten Vorgehensweisen: Diese Veränderung erfordert Zeit, Verständnis und Disziplin (Trahasch & Zimmer, 2016, S. 6). Die Struktur der Werte orientiert sich am Manifest für agile Softwareentwicklung.

1. **Unternehmensnutzen** ist wichtiger als das Festhalten an Methoden und Architekturkonzepten.

2. **Kontinuierliche Zusammenarbeit und Interaktion zwischen Anforderern und Umsetzern** sind wichtiger als Prozesse und Werkzeuge.

3. **Eingehen auf Veränderungen** ist wichtiger als Festhalten an einem Plan.

4. **Funktionierende BI-Lösungen** sind wichtiger als detaillierte Spezifikation.

Den Werten auf der linken Seite kommt eine höhere Bedeutung zu, als den Werten auf der rechten Seite. Dies bedeutet jedoch nicht, dass die Werte auf der rechten Seite unwichtig sind (Trahasch & Zimmer, 2016, S. 6).

UNTERNEHMENSNUTZEN IST WICHTIGER ALS DAS FESTHALTEN AN METHODEN UND ARCHITEKTURKONZEPTEN.

Die Herausforderung einer integrierten, unternehmensspezifischen Business Intelligence, die zum Unternehmenserfolg beiträgt, liegt in der nutzbringenden Kombination der beiden Pole standardisierter als auch agiler Lösungen. Vor dem Hintergrund des Unternehmensnutzens ist es erforderlich, agile und standardisierte Ansätze im Sinne von Architektur, Vorgehen und Organisationsformen zusammenzuführen (Trahasch & Zimmer, 2016, S. 6).

KONTINUIERLICHE ZUSAMMENARBEIT UND INTERAKTION ZWISCHEN ANFORDERERN UND UMSETZERN SIND WICHTIGER ALS PROZESSE UND WERKZEUGE.

Für die erfolgreiche Umsetzung einer BI-Lösung ist es wichtig, dass jeder seine spezifischen Fachkompetenzen in der Zusammenarbeit einbringt und fachliches Verständnis für die Gesamtlösung entwickelt. Dies erfordert die kontinuierliche Zusammenarbeit zwischen Auftraggebern, Stakeholdern und Umsetzern. Deshalb ist die direkte Interaktion zwischen den Beteiligten wichtiger als starre Prozesse und Werkzeuge (Trahasch & Zimmer, 2016, S. 6).

EINGEHEN AUF VERÄNDERUNGEN IST WICHTIGER ALS FESTHALTEN AN EINEM PLAN.

Da BI-Anwendungen häufig Änderungen unterliegen und Anforderungen sowie Rahmenbedingungen sich stetig weiterentwickeln, bringt das starre Festhalten an einem einmal ausgearbeiteten Plan nicht den größten Nutzen für das Unternehmen. Jedoch kann es ohne vertragliche Grundlage und klare Regeln in Verbindung mit einem „Big Picture“ der BI-Anwendung keine Zusammenarbeit geben. Das *„Big Picture“* sollte regelmäßig hinterfragt und gegebenenfalls angepasst werden. Das Eingehen auf Veränderungen ist wichtiger als ein Festhalten an dem Plan (Trahasch & Zimmer, 2016, S. 7).

FUNKTIONIERENDE BI-LÖSUNGEN SIND WICHTIGER ALS DETAILLIERTE SPEZIFIKATION.

Die Entwicklung eines *„Big Picture“* im Vorfeld der Implementierung und die Dokumentation von grundlegenden Architekturkomponenten ist für die Entwicklung einer BI-Lösung essentiell. Jedoch ist die Auslieferung funktionierender Lösungen wichtiger, als detaillierte Spezifikationen (Trahasch & Zimmer, 2016, S. 7).

2.2.2 Prinzipien

Die Grundwerte des Agilen Manifests bedingen eine Reihe von Prinzipien für die Entwicklung von BI Anwendungen. Diese Prinzipien werden wichtig, sobald schwierige Entscheidungen mit Zielkonflikten zu treffen sind. Die Prinzipien sind an die Prinzipien für Agile Softwareentwicklung angelehnt (Collier, 2012) (Beck, et al., 2001).

- Unsere höchste Priorität ist es, den BI-Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.

- Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des BI-Kunden.
- Liefere funktionierende BI Anwendungen regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.
- Fachexperte und Entwickler müssen während des Projekts täglich zusammenarbeiten.
- Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie Aufgaben erledigen.
- Die effizienteste und effektivste Methode Informationen an und innerhalb eines Entwicklungsteams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.
- Eine funktionierende BI-Anwendung ist das wichtigste Fortschrittsmaß.
- Wir halten Balance zwischen Projektumfang, Zeitplan und Kosten. Das Data Warehouse Team arbeitet in einem nachhaltigen Tempo.
- Ständiges Augenmerk auf technische Exzellenz und gutes Data Warehouse Design fördert Agilität.
- Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.
- In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt das Verhalten entsprechend an.

2.3 Maßnahmen zur Steigerung der BI-Agilität

Die agilen Werte um Prinzipien bilden die Grundlage für alle Methoden und Maßnahmen zur Steigerung der BI-Agilität. Wie eingangs beschrieben, handelt es sich bei BI-Agilität um die Eigenschaften einer BI-Architektur und einer BI-Aufbauorganisation sowie der zugehörigen BI-Prozesse (Krawatzeck, Zimmer, & Trahasch, 2013, S. 59). Architektonische Ansätze für einen hohen Grad an BI-Agilität stellen zum Beispiel Sand-Boxes für den Fachbereich, in Verbindung mit geregelten Prozessen zur Weiterentwicklung des Data Warenhauses (DWH), dar. Verschiedene Möglichkeiten die BI-Agilität mit Hilfe der Architektur zu unterstützen, werden in (Trahasch & Zimmer, 2016) beschrieben. Organisatorische Maßnahmen können neben architektonischen Maßnahmen zur BI-Agilität beitragen. Eine organisatorische Möglichkeit ist ein zentral organisiertes Analystenteam, dass Ad-hoc-Anfragen bearbeitet. Innerhalb des architektonischen und organisatorischen Rahmens können geeignete Vorgehensmodelle die BI-Agilität zusätzlich erhöhen. Ein Vorgehensmodell liefert Rahmenbedingungen für die Produktentwicklung, indem es Rollen, Verantwortlichkeiten, Abläufe und die zu erstellenden Artefakte definiert (Krawatzeck, Zimmer, & Trahasch, 2013, S. 59).

Technische Maßnahmen

Agile Methoden lassen sich auf verschiedene Weisen auf den BI-Bereich übertragen und können Anhand der Entwicklungsphasen des Wasserfallmodells kategorisiert werden (Krawatzeck, Zimmer, & Trahasch, 2013, S. 59 f.).

- Designphase – Methoden, die flexible Architekturen ermöglichen
- Entwicklungsphase – Methoden, die während der Produktentwicklung angewendet werden
- Bereitstellungsphase – Methoden zur kontinuierlichen Produkterstellung

In der Designphase lässt sich Agilität beispielsweise durch modellgetriebene Softwareentwicklung erhöhen. Verschiedene Methoden der Qualitätssicherung, die es erlauben flexibel auf Anforderungen zu reagieren, sind Pair Programming oder Test Driven Development (TDD). Als Maßnahme für die kontinuierliche Bereitstellung dienen Continuous Integration (CI) oder Continuous Delivery. „All diese Methoden führen jedoch nur dann in einem angemessenen Zeitrahmen zu zuverlässigen Ergebnissen, wenn nach Änderungen die notwendige Qualitätsprüfung zur Gewährleistung der Korrektheit von vorab vorhandenen und neuen Funktionalitäten zum Großteil mit automatisierten Tests durchgeführt wird.“ (Trahasch & Zimmer, 2016, S. 6)

Organisatorische Maßnahmen

Organisatorische Maßnahmen stellen einen wichtigen Einflussfaktor auf BI-Agilität dar. Eine enge Verzahnung der Fach- und IT-Abteilung sowie eine im Unternehmen verankerte Feedbackkultur, sind Beispiele für solche organisatorischen Maßnahmen. In allen methodischen Ansätzen zur Steigerung der Agilität kommt der direkten Zusammenarbeit des Fachbereichs mit dem BI-Entwicklungsteam eine wichtige Rolle zu. Um die direkte Kommunikation zu fördern, sollte die räumliche Trennung von Projektbeteiligten aufgehoben werden (Trahasch & Zimmer, 2016, S. 13). „Eine kundenorientierte Ausrichtung und eine offene Kommunikation sowie die mit BI-Agilität einhergehende Möglichkeit, flexibel, effizient und zeitnah auf Anforderungen zu reagieren, ermöglicht es Unternehmen, die Softwarequalität zu steigern, Entscheidungen schneller treffen zu können, sich besonders am Markt zu positionieren und einen früheren Return on Investment zu erzielen.“ (Trahasch & Zimmer, 2016, S. 14)

3 BI – ARCHITEKTUR

Die IT-Unterstützung des Managements wird Business-Intelligence (BI) genannt. *„'Intelligence' ist Wissen, das durch die Erfassung, Integration, Transformation Speicherung Analyse und Interpretation geschäftsrelevanter Informationen generiert wird."* (Hansen & Neumann, 2009) Konzepte des Data-Warehouse, OLAP, und Data Mining werden unter diesem Begriff diskutiert. Sie bilden ein logisches Pendant zu den operativen Systemen, wie zum Beispiel betriebswirtschaftlicher Standardsoftware (Hahne, 2014, S. 1). Durch die zunehmend strategische Ausrichtung der Informationsverarbeitung erhalten diese Konzepte einen neuen Stellenwert in der Praxis. Unternehmen, die in der Lage sind, durch den Einsatz innovativer Technologien schnell und flexibel auf sich ändernde Marktfaktoren und Kundenbedürfnisse zu reagieren, können so einen Vorsprung gegenüber Wettbewerbern sichern. Die Anforderung besteht darin, die in den operativen Systemen vorliegenden Daten in geeigneter Form und zum richtigen Zeitpunkt für Analysen bereitzustellen und für die Entscheidungsfindung nutzbar zu machen (Gabriel, Gluchowski, & Pastwa, 2009, S. 9). Aus diesen Anforderungen folgt das Ziel, aus den Datenbeständen personen-, problem- und situationsgerechte Informationen bereitzustellen. Gabriel, Gluchowski, & Pastwa führen aus diesem Grund neben dem Begriff des analyseorientierten Informationssystems den Begriff der Business-Intelligence-Systeme ein, die diese Aufgabe bewältigen sollen (Gabriel, Gluchowski, & Pastwa, 2009, S. 4).

3.1 BI- Hub-and-Spoke-Architektur

Diese Anforderungen verlange nach einer klaren Architektur sowie adäquater Methoden des Entwurfs. Bekannte Architekturvarianten unterscheiden sich deutlich hinsichtlich der Anzahl der Komponenten und der gesamten Komplexität. Eine Architekturvariante, die die Anforderungen nach dynamischer Entwicklung einerseits und effizienten Betrieb andererseits in Einklang bringt, ist die Hub-and-Spoke-Architektur (Göhl & Hahne, 2012). Abb. 3-1 zeigt die Architekturkomponenten und Datenflüsse einer klassischen BI- Hub-and-Spoke-Architektur. In einer Hub-and-Spoke-Architektur dient das Core Data-Warehouse als Hub und erfüllt die Aufgaben der Integration, Qualitätssicherung und Datenverteilung an die Data Marts als Spokes, die einen hohen Grad an Anwendungsorientierung und vordefinierten betriebswirtschaftlichen Anreicherungen und Aggregationen aufweisen (Kemper, Baars, & Mehanna, 2010).

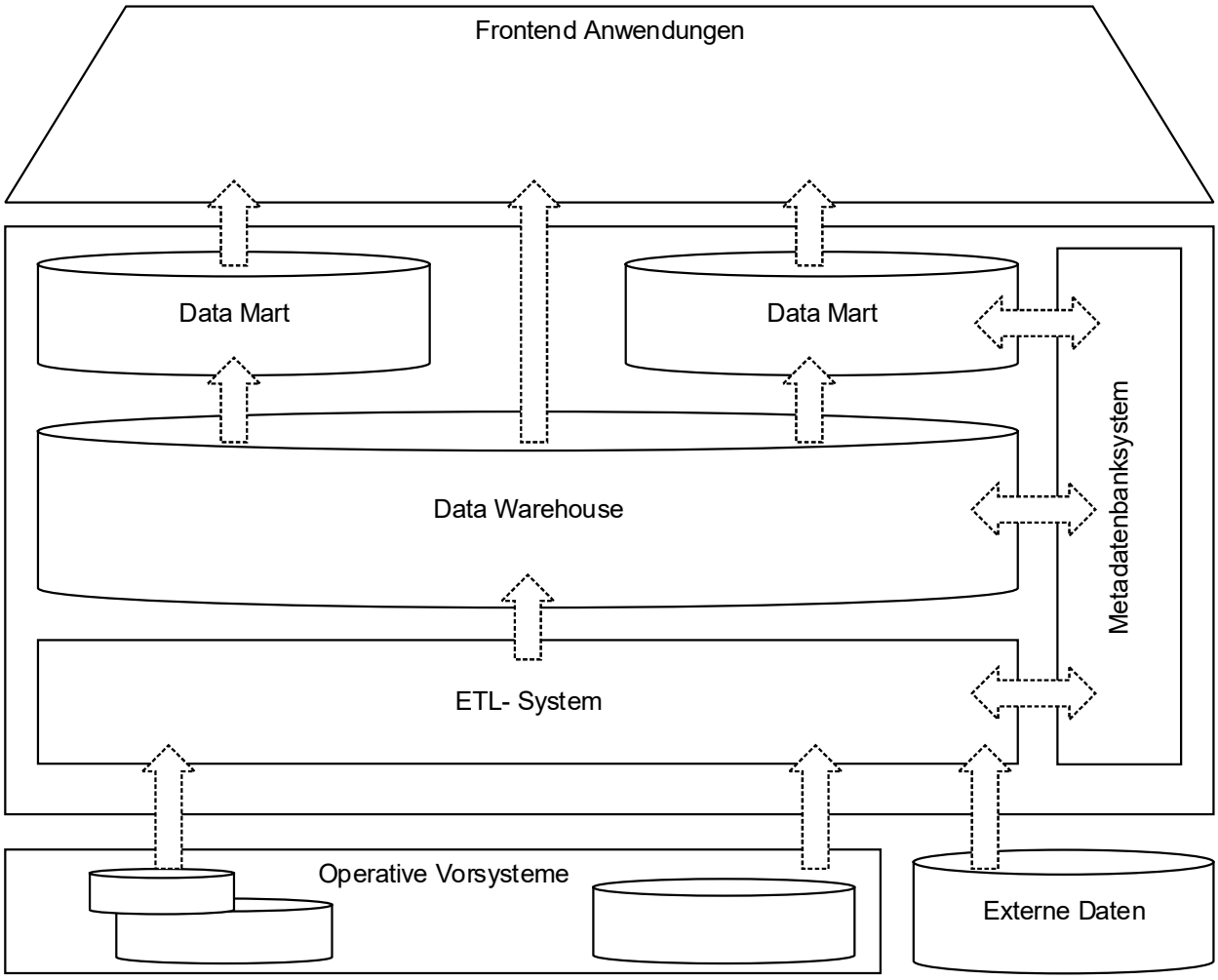

Abb. 3-1 Die Architekturkomponenten und Datenflüsse einer klassischen BI-Hub-and-Spoke-Architektur (Zimmer, 2015)

3.2 Datenquellen

Datenquellen dienen als Ausgangspunkt eines jeden Data-Warehouse-Projekts. Mit ihnen beginnt der Prozess des Data Warehousing, und ihre Beschaffenheit sowie Qualität haben einen unmittelbaren Einfluss auf das Analyseergebnis, welches Ziel des Prozesses ist. Auswahl und Sicherung der Datenqualität sind damit für ein erfolgreiches Data-Warehouse-Projekt von entscheidender Bedeutung. In der Regel dienen die operativen, transaktionalen Systeme einer Unternehmung als Datenquelle (Bauer & Günzel, 2013, S. 40).

3.3 Extraktion

Aktualisierungen der Quellsysteme werden inkrementell in den bestehenden Datenbestand des Data-Warehouse integriert, sodass der Datenbestand stets aktuell gehalten wird. Ein vollständiges Löschen inklusive anschließender gesamtheitlicher Neubeladung ist aus zweierlei Gründen

nicht praktikabel: Zum einen sind die zu verarbeitenden Datenvolumina der Vorsysteme typischerweise so groß, dass dies nicht in einem akzeptablen Zeitraum zu bewerkstelligen wäre und zum anderen würde die Historisierung der Daten im Data-Warehouse auf diese Weise verloren gehen (Bauer & Günzel, 2013, S. 49). Der Zeitpunkt, an dem die Daten übertragen werden, hängt maßgeblich von der Semantik der Daten bzw. von den durchzuführenden Auswertungen ab (Bauer & Günzel, 2013, S. 51). Häufig werden Tages-, Wochen- oder Monatsgrößen gefordert. Um die operativen Systeme nicht zu beeinträchtigen, findet die Übertragung oftmals nachts oder am Wochenende statt (Gabriel, Gluchowski, & Pastwa, 2009).

3.4 ETL

ETL steht für Extraktion, Transformation und Laden. Damit wird die Überführung von Daten zwischen den verschiedenen transaktionalen Informationssystemen und den Data-Warehouse-Datenhaltungskomponenten organisiert. Somit ist die Datenbewirtschaftung mit ETL ein wesentlicher Bestandteil fast jedes Data-Warehouse-Projektes (Bange, 2006, S. 92).

3.5 Staging Area

In der Staging Area werden die zuvor extrahierten Quelldaten temporär zwischengespeichert. Bei der Übertragung findet meist noch keine Transformation der Daten statt, sodass sich in der Persistent Staging Area (PSA) zunächst eine 1:1-Kopie der Quelldaten befindet. Durch die Entkoppelung von den Quellsystemen besteht im weiteren Verlauf des ETL-Prozesses nicht mehr das Risiko, die Quellen in ihrem laufenden Betrieb zu beeinflussen. Die gewünschten Transformationen können nun auf der Datenbasis der Staging Area durchgeführt werden (Bauer & Günzel, 2013). Die Staging Area dient häufig als Bereich zur Zwischenspeicherung von bspw. tagesaktuellen Daten. Die extrahierten Daten werden über einen entsprechenden Zeitraum gesammelt (bspw. bis eine Woche abgeschlossen ist), um sie anschließend auf der gewünschten Aggregationsebene zu verdichten. Beispielhaft wären so keine tagesaktuellen, sondern lediglich wochenaktuelle Daten im Data-Warehouse enthalten. Die Funktion dieser Komponente besteht folglich darin, einen bestimmten Zeitpunkt bzw. Stand der Quelldaten bis zu ihrer weiteren Verarbeitung festzuhalten (Bauer & Günzel, 2013).

3.6 Transformation

Operative respektive transaktionale Daten sind auf die Anforderungen der mengen- und wertorientierten Abrechnungs-, Administrations- und Dispositionssysteme ausgerichtet. Sie gewährleisten einen reibungslosen Ablauf des Tagesgeschäfts. Daher stehen bei der Konzeption operativer

Datenhaltungssysteme keine Anforderungen hinsichtlich der Analysefähigkeit der Systeme im Fokus. Aufgrund dessen müssen die operativen Daten, die in der Staging Area vorgehalten werden, im nächsten Schritt derart transformiert werden, dass aus ihnen management- und entscheidungsrelevante Informationen gewonnen werden können. Diese hierfür notwendigen Prozesse werden unter dem Obergriff der Transformation zusammengefasst. Dieser setzt sich nach Kemper aus den Sub-Prozessen der Filterung, Harmonisierung, Aggregation und Anreicherung zusammen (Kemper & Finger, 2006, S. 115).

3.7 Laden

Nachdem die Daten transformiert wurden, müssen diese in den bestehenden Datenbestand integriert werden. Dieser Schritt erfolgt typischerweise zu einem oder mehreren festgelegten Zeitpunkten oder Ereignissen. In der Regel werden die Aktionen Transformation und Laden in einem Schritt durchgeführt, sodass die transformierten Daten nicht zunächst zwischengespeichert werden müssen, sondern direkt in die Zieltabelle geschrieben werden können. Sofern bereits bei der Extraktion berücksichtigt, werden in diesem Schritt nur neue oder geänderte Daten übertragen, sodass die benötigten Rechenressourcen für die Integration möglichst gering ausfallen. Bei der Integration der Daten muss das Historisierungskonzept des Data-Warehouse berücksichtigt werden. Geänderte Datensätze dürfen nicht einfach überschrieben, sondern müssen als zusätzliche Datensätze unter Verwendung eines Änderungszeitstempels abgelegt werden. Da der Transformations- und Beladungsvorgang häufig mit einer zusätzlichen Belastung der beteiligten Systeme einhergeht, erfolgen diese in der Regel zu Zeiten außerhalb der Arbeitszeiten, wie z. B. nachts oder am Wochenende (Bauer & Günzel, 2013, S. 98).

3.8 Zentrale Datenbereitstellung / Data-Warehouse

Die zentrale Stelle der Datenbereitstellung wird in der Regel als Data-Warehouse bezeichnet, jedoch finden auch die Bezeichnungen Core Data-Warehouse oder Enterprise Data-Warehouse häufig Verwendung. In einer Hub-and-Spoke-Architektur dient das Core Data-Warehouse als Hub und erfüllt die Aufgabe der Integration, Qualitätssicherung und Datenverteilung an die Data Marts als Spokes. Die redundanzfreie Logik und die zentrale Datenintegration und Aufbereitung sind Vorteile dieses Ansatzes (Trahasch & Zimmer, 2016, S. 63).

3.9 Spezialisierte Datenbereitstellung/Data Mart

Data Marts weisen einen hohen Grad an Anwendungsorientierung und vordefinierten betriebswirtschaftlichen Anreicherungen und Aggregationen auf. Die Verlagerung von Businesslogik

möglichst nah zum Endbenutzer, hat sich auf Grund der großen Volatilität von geschäftlichen Anforderungen bewährt. Das späte Ansiedeln dieser Geschäftsregeln im Prozess der Datenveredelung führt dazu, dass Änderungen nicht das Core Data-Warehouse betreffen (Trahasch & Zimmer, 2016, S. 65). Durch die logische Entkopplung der Data-Marts und der geringen Abhängigkeit untereinander eignen sich Data-Marts besonders für Agile BI-Ansätze.

4 TESTEN

"Testing can be used to show the presence of bugs, but never show their absence! “

Edsger W. Dijkstra

4.1 Begriffseinordnung

4.1.1 Fehler

Ein Fehlverhalten von Computerprogrammen wird im Allgemeinen als Softwarefehler bezeichnet. Die Definition eines Softwarefehlers nach IS0 9000:2005 ist angelehnt an die allgemeine Definition eines Fehlers, der „Nichterfüllung einer Anforderung“. Demnach ist ein Softwarefehler als „Abweichung des IST (beobachtete, berechnete Zustände oder Vorgänge) vom SOLL (festgelegte korrekte Zustände und Vorgänge), wenn sie die vordefinierte Toleranzgrenze (die auch 0 sein kann) überschreitet“ (ISO/IEC, ISO 9000:2015, 2015) definiert. Der in dieser Arbeit verwendete Fehlerbegriff stammt von (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010) und baut auf der Definition des ISTQB (ISTQB, 2017) auf, wonach der Fehler als Teil einer Ursachenkette definiert ist:

> *„Ein Fehler ist [...] die Nichterfüllung einer festgelegten Anforderung, eine Abweichung zwischen dem Istverhalten (während der Ausführung des Tests oder des Betriebs festgestellt) und dem Sollverhalten (in der Spezifikation oder Anforderung festgelegt).“*
> (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 7)

Da im Gegensatz zu physikalischen Systemen Fehler in einem Softwaresystem nicht durch Alterung oder Verschleiß entstehen, sondern jeder Fehler oder Mangel seit dem Zeitpunkt der Entwicklung in der Software vorhanden ist, wird zwischen dem Auftreten und der Ursache eines Fehlers unterschieden. Das Auftreten des Fehlers wird als Fehlerwirkung bezeichnet. Beim Ausführen oder Testen der Software wird eine Fehlerwirkung nach außen sichtbar. Eine Fehlerwirkung hat ihren Ursprung in einem Fehlerzustand. Ein Fehlerzustand ist ein „Defekt (innerer Fehlerzustand) in einer Komponente oder einem System, der eine geforderte Funktion des Produkts beeinträchtigen kann, z. B. inkorrekte Anweisung oder Datendefinition. Ein Fehlerzustand, der zur Laufzeit angetroffen wird, kann eine Fehlerwirkung einer Komponente oder Systems verursachen.“ (ISTQB, 2017) Ein Fehlerzustand muss jedoch nicht zum Tragen kommen. Der Umstand, bei dem ein Fehlerzustand die Aufdeckung eines anderen verhindert, wird als Fehlermaskierung bezeichnet (ISTQB, 2017) (nach IEEE 610). Eine Fehlerwirkung tritt in diesem Fall erst dann zutage, nachdem der oder die maskierenden Fehlerzustände korrigiert worden sind. Für das Vorliegen eines Fehlerzustandes können Fehlhandlungen einer Person aber auch Faktoren aus dem

Umfeld - wie Strahlung, Magnetismus mit entsprechender Auswirkung auf die Hardware usw. - verantwortlich sein.

4.1.2 Testen

Das Entstehen von Fehlern zu verhindern und entstandene Fehler möglichst früh zu entdecken, ist Ziel der Qualitätssicherung. Es wird zwischen Maßnahmen zur konstruktiven Qualitätssicherung und Maßnahmen zur analytischen Qualitätssicherung unterschieden. Die konstruktive Qualitätssicherung setzt auf Maßnahmen während der Entwicklung. Sie basiert auf dem Einsatz von Methoden, Konstruktionsprinzipien, formalen Verfahren, Entwicklungswerkzeugen und Vorgehensmodellen. Entstandene Fehler möglichst früh zu entdecken und die erreichte Qualität zu bewerten, ist das Ziel analytischer Qualitätssicherung. Testen ist Teil der analytischen Qualitätssicherung. Ein Softwaretest prüft und bewertet Software auf Erfüllung der für ihren Einsatz definierten Anforderungen und misst ihre Qualität. Durch das Testen verringern sich die Risiken beim Einsatz der Software, da Fehlerwirkungen während des Testens gezielt und systematisch aufgedeckt werden. Tests während der Softwareentwicklung dienen dazu, die Software möglichst fehlerfrei in Betrieb zu nehmen (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 6 ff.) Ernst Denert definiert den Test als einen überprüfbaren und jederzeit wiederholbaren Nachweis der Korrektheit eines Softwarebausteins – relativ zu vorher festgelegten Anforderungen. Die Definition des ISTQB (ISTQB, 2017) meint mit Testen explizit die Ausführung der Software*:* „the process of operating a system or component under specified conditions, observing or recording the results and making an evaluation of some aspects of the system or component*“* Das ISTQB erweitert den Testbegriff auf „alle Aktivitäten des Lebenszyklus […] mit dem Ziel, sicherzustellen, dass die Arbeitsergebnisse der Softwareentwicklung allen festgelegten Anforderungen genügen:

> *„Der Prozess, der aus allen Aktivitäten des Lebenszyklus besteht (sowohl statisch als auch dynamisch), die sich mit der Planung, Vorbereitung und Bewertung eines Softwareprodukts und dazugehöriger Arbeitsergebnisse befassen. Ziel des Prozesses ist sicherzustellen, dass diese allen festgelegten Anforderungen genügen, dass sie ihren Zweck erfüllen, und etwaige Fehlerzustände zu finden“. (ISTQB, 2017)*

Demnach umfasst das Testen nicht nur das Ausführen des Prüfgegenstandes, sondern auch Tätigkeiten, die den Prüfgegenstand nicht ausführen. Statische Tests führen den Prüfgegenstand nicht aus, sind also auf alle Entwicklungsprodukte anwendbar (Winter, Roßner, Brandes, & Goetz, 2016, S. 16). Als dynamischer Test wird, „[…] jede (im Allgemeinen stichprobenartige) Ausführung eines Testobjekts […], die der Überprüfung des Testobjekts dient“ (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 9) bezeichnet.

Testen verfolgt mehrere Ziele (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 9):

- Ausführung des Programms mit dem Ziel, Fehlerwirkungen nachzuweisen.
- Ausführung des Programms mit dem Ziel, die Qualität zu bestimmen.
- Ausführung des Programms mit dem Ziel, Vertrauen in das Programm zu erhöhen.
- Analysieren des Programms oder der Dokumente, um Fehlerwirkungen vorzubeugen.

Das Testobjekt wird mit Eingabedaten versehen und zur Ausführung gebracht. Wenn kein ablauffähiges Programm vorliegt, wie zum Beispiel in den unteren Teststufen (Komponenten- und Integrationstest), wird das Testobjekt in einen Testrahmen eingebettet (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 109). Abb. 4-1 zeigt diesen Zusammenhang.

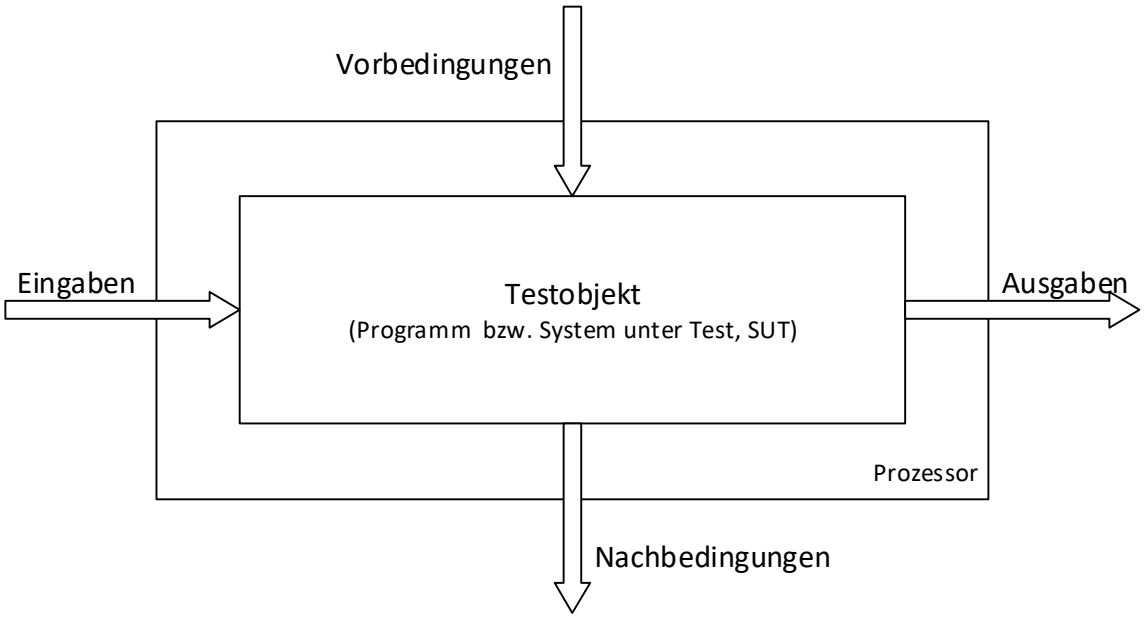

Abb. 4-1: Testrahmen (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 109)

4.2 Testprozess

Für die Einordnung des Testens in den gesamten Entwicklungsablauf wird ein Ablaufplan für die Testarbeiten selbst benötigt. Die Arbeitsschritte werden in die Aufgaben Testplanung & Steuerung, Testanalyse und –design, Testrealisierung und Durchführung, Testauswertung und Bericht sowie Abschluss der Testaktivitäten gegliedert. Die Aufgaben des Testprozesses werden dem gewählten Vorgehensmodell entsprechend in das Projekt integriert. Die Phasen des Testprozesses sind parallel zu den Phasen des Entwicklungsprozesses zu sehen (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 2).

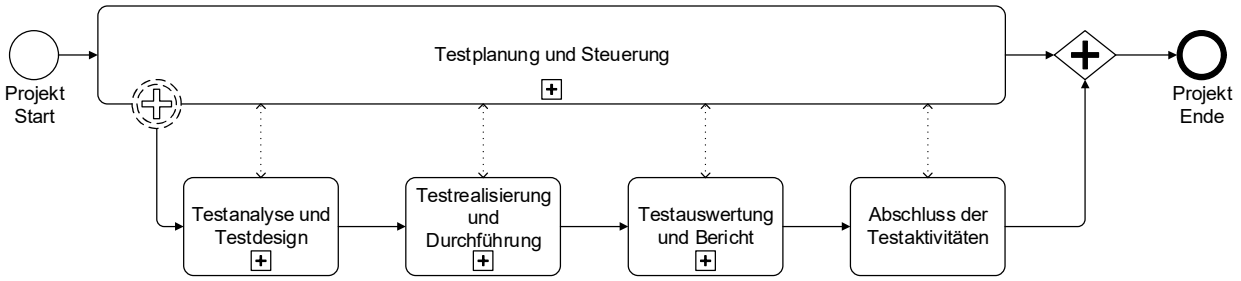

Abb. 4-2: Fundamentaler Testprozess (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 20)

4.2.1 Testplanung

Während der Testplanung werden die Ziele und Vorgehensweisen des Testens festgelegt und die Ressourcen kalkuliert. Kernaufgabe der Planung ist die Bestimmung der Teststrategie. Die Teststrategie definiert welche Testobjekte mit welchen Testverfahren und welchem Umfang getestet werden. Je nach zu erwartendem Risiko und schwere der Auswirkungen beim Eintreten der Fehlerwirkung sind die Testaktivitäten auf die einzelnen Systemteile zu verteilen. Kritische Systemteile müssen eine höhere Aufmerksamkeit im Testen erfahren. Das Testkonzept dient als „Projekthandbuch des Testens“ und ist Basis für die Teststeuerung. Die Aktualisierung der Testplanung ist ein kontinuierlicher Prozess. Anpassungen der Planung sind Ergebnis fortwährender Überwachung der aktuellen Testaktivitäten im Vergleich zur Planung. Für Testautomatisierung spielt die Planung eine zentrale Rolle. Viele Überlegungen müssen schon vorab angestellt werden und sind wesentlicher Bestandteil der Teststrategie (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 21 ff.). Es muss entschieden werden, in welchen Teststufen und in welchem Ausmaß Testautomatisierung eigesetzt werden soll. Testautomatisierung ist in allen Teststufen möglich (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 11 f.)

4.2.2 Testanalyse und Entwurf

In der Analyse- und Entwurfsphase wird mit der Testfallerstellung anhand der Informationen aus dem Testkonzept und den Spezifikationen begonnen. Voraussetzung für den Testentwurf ist eine vorhandene Testbasis. Die Testbasis ist die Grundlage für die Erstellung der Tests. Anforderungsdokumenten, Architekturdokumenten, Ergebnissen der Risikoanalyse oder auch weitere Dokumente, die im Rahmen der Softwareentwicklung erstellt bzw. genutzt werden, können Basis für den Testentwurf sein. Zudem muss das Testobjekt gewissen Anforderungen genügen, die bereits beim Entwurf und der Entwicklung des Testobjekts zu beachten sind. Die wichtigsten Faktoren sind die Ansprechbarkeit der Schnittstellen und die Modularität des Testobjekts. Testobjekt und Testziele werden durch Anwendung definierter Testentwurfsverfahren zu Testfällen detailliert. Die systematische Anwendung von Testentwurfsverfahren stellt sicher, dass „Testfälle für

verschiedene Testobjekte unterschiedlichen Entwicklungsständen projektübergreifend und nachvollziehbarer Weise und gleichbleibender Qualität entstehen.“ Das Ergebnis ist die Testspezifikation. Es ist darauf zu achten, dass eine eindeutige Verbindung zwischen den zu testenden Anforderungen und den spezifizierten Testfällen besteht. Es muss klar sein welche Testfälle welche Anforderungen prüfen und umgekehrt.

Die Spezifikation erfolgt in zwei Schritten: Es werden logische Testfälle definiert die anschließend konkretisiert werden. Die Auswahl der logischen Testfälle zu jeder Testmethode erfolgt auf Grundlage der jeweiligen Testbasis. Die Spezifikation der Testfälle kann zu unterschiedlichen Zeitpunkten im Softwareentwicklungsprozess erfolgen. Werden die Testfälle aus der Spezifikation der Testobjekte ermittelt, wird von Blackbox-Verfahren gesprochen. Testfälle auf Basis des Quellcodes heißen Whitebox-Verfahren.

Für jeden Testfall muss die Ausgangssituation beschrieben werden. Es muss klar sein welche Grundbedingungen für den Test gelten. Zudem ist festzulegen welches Testergebnis bzw. welches Verhalten erwartet wird. Zu den Ergebnissen gehören neben den Ausgaben auch Änderungen an globalen Daten und Zuständen sowie alle weiteren Veränderungen als Reaktion auf die Durchführung des Testfalls. Die zu erwartenden Sollergebnisse werden mit Hilfe eines Testorakels bestimmt. Als Testorakel kann beispielsweise die Spezifikation dienen. Eine andere Möglichkeit ist die Umkehrfunktion: Die Ausgabe des Tests wird mit der ursprünglichen Eingabe verglichen. (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 23 ff.)

In dieser Phase beginnt auch die Umsetzung der Testfälle in der Testautomatisierung. Die Testumgebung wird vorbereitet und zu automatisierende Testfälle ausgewählt. Die Auswahl wird anhand der Vorgaben aus dem Testkonzept getroffen. Hier ist eine Prüfung auf technische Machbarkeit notwendig. Nicht immer sind für die Automatisierung vorgesehene Testfälle, auch technisch mit vertretbarem Aufwand umsetzbar. Bei der Auswahl ist eine breite Testabdeckung einer großen Testtiefe vorzuziehen (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 16).

4.2.3 Testrealisierung und Testdurchführung

Im Rahmen der Testrealisierung werden logische Testfälle mit Testdaten hinterlegt und so zu konkreten Testfällen. Diese werden in Testszenarien gruppiert und der Ablauf anhand des Testkonzeptes festgelegt. Voraussetzung für die Durchführung der Testfälle ist die Realisierung der Testinfrastruktur und des Testrahmens. Wenn die definierten Vorbedingungen für die Testfälle erfüllt sind, kann, sobald das Testobjekt zur Verfügung steht, mit der Abarbeitung der Testfälle begonnen werden. Neben der Definition der Testfälle ist es notwendig den konkreten Ablauf der Tests festzulegen. Hierbei ist die Priorität der Testfälle zu berücksichtigen. Die Ausführung der Testfälle kann manuell oder durch Werkzeuge erfolgen. Tritt bei der Ausführung ein Unterschied

zwischen dem tatsächlichen und dem erwartenden Ergebnis auf, ist bei der Auswertung der Testprotokolle zu entscheiden, ob tatsächlich eine Fehlerwirkung vorliegt. Liegt eine Fehlerwirkung vor, ist diese zu dokumentieren und eine grobe Prüfung der Ursachen durchzuführen. (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 27 ff.)

Sobald das Testobjekt korrigiert wurde, werden entsprechende Retests und Regressionstests durchgeführt. Sollte die Zeit nicht für alle Testfälle ausreichen, so ist eine sinnvolle Auswahl zu treffen. Hierzu ist eine Priorisierung der Testfälle erforderlich. Die Priorisierung kann zum Beispiel anhand von risikoorientierter Testauswahl geschehen (0).

Während im manuellen Test die Testdaten direkt hinterlegt werden, und somit eine konkrete Ausprägung des Testfalls entsteht, werden Testdaten beim automatisierten Test oftmals erst dynamisch zur Laufzeit ermittelt. Testfälle müssen während der Implementierung dahingehend geprüft werden, ob sie ihren Zweck erfüllen und funktionieren. Die Möglichkeit der Rückverfolgbarkeit Abb. 4-3: Rückverfolgbarkeit von Testfällen Abb. 4-3) zu den logischen Testfällen und den Anforderungen im Sinne vom Impact- und Traceability-Analysen ist, teilweise auch aus regulatorischer Sicht, notwendig (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 17).

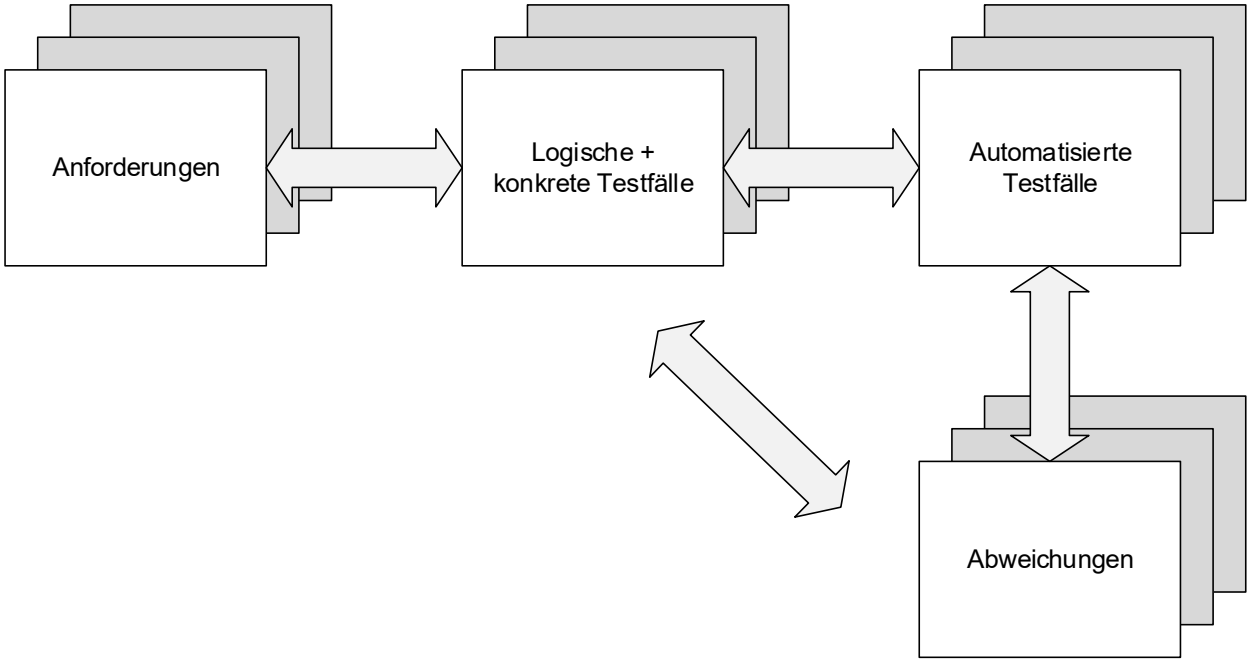

Abb. 4-3: Rückverfolgbarkeit von Testfällen (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 17)

VORBEREITEN DES TESTABLAUFS

Testfälle werden in sinnvolle Gruppierungen zusammengefasst um einen effizienten Testablauf zu gewährleisten. Dies betriff die Testdaten, aber auch die funktionale und logische Zusammengehörigkeit der Testfälle. Tabelle 4.1 zeigt eine ungünstige Ablaufzusammenstellung von Testfällen. Im ersten Schritt muss der Benutzer >>Maier<< zunächst angelegt werden, bevor er während

der Testdurchführung wieder gelöscht wird. Vor dem zweiten Ablaufschritt muss der Benutzer erneut angelegt werden, damit Änderungen am Benutzer vorgenommen werden können (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 18).

Ablaufschritt	Testfall	Beschreibung
1	TF-2602	Benutzer >>Maier<< löschen
2	TF-1405	Benutzer >>Maier<< ändern
3	TF-2102	Benutzer >>Maier<< anlegen

Tabelle 4.1: Ungünstige Anordnung von Testfällen (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 26)

Tabelle 4.2 beschreibt einen effizienteren Ablauf, in dem die Datenbasis nicht nach jedem Testfall zurückgesetzt werden muss.

Ablaufschritt	Testfall	Beschreibung
1	TF-2102	Benutzer >>Maier<< anlegen
2	TF-1405	Benutzer >>Maier<< ändern
3	TF-2602	Benutzer >>Maier<< löschen

Tabelle 4.2 Günstige Anordnung von Testfällen (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 18)

4.2.4 Abschluss der Testaktivitäten

Die rückblickende Betrachtung der Testaktivitäten dient dem Lerneffekt für nachfolgende Projektphasen, aber auch der Überprüfung von gesteckten Zielen. Zudem werden alle Testmittel mit dem Ziel archiviert, die gleichen Tests zu einem späteren Zeitpunkt erneut durchführen zu können (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 21)

4.3 Teststufen

4.3.1 Das V-Modell

Die Grundidee des allgemeine V-Modells ist, dass Entwicklungsarbeiten und Testarbeiten zueinander korrespondieren. Immer detaillierter werdende Entwicklungsschritte korrespondieren mit den Integrations- und Testarbeiten, in deren Verlauf elementare Programmbausteine sukzessive zu größeren Teilsystemen zusammengesetzt und jeweils auf richtige Funktionalität geprüft werden. Folgende Prinzipien stehen hinter dem V-Modell:

- Konstruktions- und Testaktivitäten tragen gleichzeitig zum Produkt bei

- Tests einer Stufe prüfen das Produkt auf einer ganz bestimmten Abstraktionsebene
- Prüfungen können den Charakter einer Verifizierung oder einer Validierung besitzen

Die konstruktiven Aktivitäten lassen sich in Abstraktionsstufen Anforderungsdefinition, funktionaler Systementwurf, technischer Systementwurf, Komponentenspezifikation und Programmierung einteilen. Korrespondierend ist jeder Abstraktionsstufe eine Teststufe zugeordnet. Abb. 4-4 zeigt diese Zusammenhänge: Die einzelnen Softwarebausteine werden zunächst im Komponententest hinsichtlich der Vorgaben der Spezifikation überprüft. Der Integrationstest prüft, ob die Interaktion von Komponentengruppen wie im technischen Systementwurf funktioniert. Der Systemtest prüft, ob das System als ganzes die spezifizierten Anforderungen erfüllt. Abschließend wird das System im Abnahmetest aus Kundensicht auf die vertraglich vereinbarten Leistungsmerkmale geprüft (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 43).

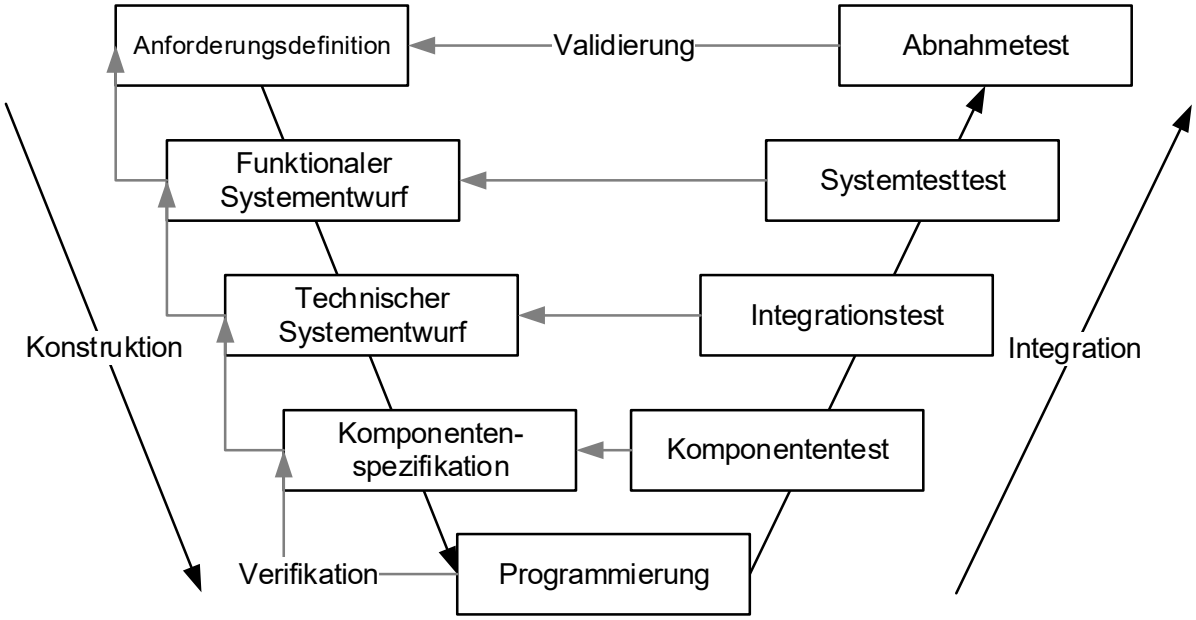

Abb. 4-4: Allgemeines V-Modell (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 3)

Die zeitliche und inhaltliche Ausgestaltung dieser Phasen ergibt sich aus dem gewählten Softwareentwicklungsmodell bzw. dem Projektvorgehen. Wobei auch bei agilen Testpraktiken die Teststufen dem aus dem V-Modell entsprechen. Diese Teststufen werden benötigt, da sie unterschiedliche Ziele verfolgen, bei denen unterschiedliche Testmethoden und unterschiedliche Testwerkzeuge zum Einsatz kommen. In einem agilen Projektvorgehen laufen die Teststufen parallel innerhalb jedes Sprints, im Idealfall täglich ab. Die Testpyramide (Cohn, 2009) ist ein Konzept bzw. eine Metapher, die dem Scrum-Team hilft, zu überprüfen, ob die vorhandenen Testfälle angemessen über sämtliche Teststufen verteilt sind (Linz, 2017, S. 47).

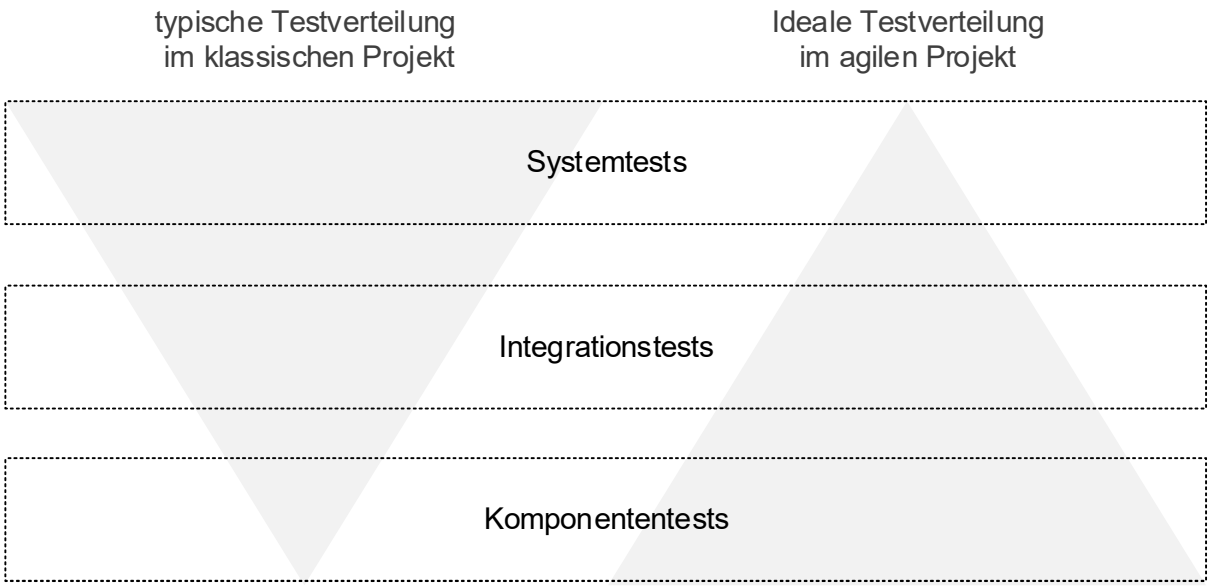

Abb. 4-5: Testpyramide im klassischen vs. agilen Projekt (Cohn, 2009)

Automatisierte Unit Tests liefern den Programmierern frühestmögliches Feedback zu jeder Codeänderung. Daher wird ein agiles Team möglichst viele automatisierte Unit Tests erstellen, pflegen und einsetzen können. Integrationstests ergänzen Unit Tests. Sie können wie Unit Tests automatisiert erstellt und automatisiert ausgeführt werden. Die Testumgebung ist jedoch in der Regel komplexer und ihre Laufzeit ist im Schnitt höher als die der Unit Tests. Die Anzahl sollte also geringer sein als die der Unit Tests. Tests auf Systemebene sind am aufwendigsten, sowohl was ihre Erstellung, Automatisierung als auch ihre Durchführung betrifft. Aus diesen Überlegungen ergibt sich die gezeigte, anzustrebende Pyramide (Linz, 2017, S. 47).

4.3.2 Komponententest

Eine Komponente ist die „kleinste Softwareeinheit, die für sich getestet werden kann“ (ISTQB, 2017). Als Komponententest (Unit Test) werden somit alle dynamischen Tests, die einen einzelnen Softwarebaustein prüfen, bezeichnet. Dies soll bausteinexterne Einflüsse ausschließen, sodass sich eine Fehlerwirkung explizit dem getesteten Baustein zuordnen lässt. Als Basis können demgemäß die komponentenspezifischen Anforderungen und das Softwaredesign der Komponente herangezogen werden. Als Basis für Whitebox – Testfälle oder für eine Aussage zur Testabdeckung, kann zudem der Source Code einer Komponente analysiert werden. Typische Testobjekte sind Programmmodule, (Datenbank-) Skripte und andere Softwarebausteine. Als wichtigste Aufgabe des Komponententests beschreiben Spillner & Linz „die Sicherstellung, dass das jeweilige Testobjekt die laut seiner Spezifikation geforderte Funktionalität korrekt und vollständig realisiert.“ Als Funktionalität wird hierbei das Ein-/Ausgabeverhalten des Testobjekts gesehen. Jeder Testfall deckt eine bestimmte Teilfunktionalität des Objektes ab (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 44 ff.).

4.3.3 Integrationstest

Als Integration werden zu größeren Baugruppen bzw. Teilsystemen verbundene Komponenten bezeichnet. Der Integrationstest prüft, ob die verbundenen Einzelteile im Zusammenspiel funktionieren. Ziel ist es Fehlerzustände in Schnittstellen und im Zusammenspiel zwischen integrierten Komponenten zu finden. Der Test überprüft den Datenaustausch der Bausteine. Die Testfälle müssen die Datenpakete bzw. Nachrichten, die ausgetauscht werden können, möglichst umfassend abdecken. Die Bausteine werden zum Test miteinander verbunden. Zudem werden auch Schnittstellen zu anderen bereits vorhandenen Systemen, oder Systemkomponenten, die auf Standardkomponenten basieren, getestet. Wenn ein Fehler aufgedeckt wird, so kann dessen Ursache in einem der beiden Bausteine liegen oder auch im Übertragungsweg.

Als Testbasis können demnach z.B. das Software- und Systemdesign bzw. die Systemarchitektur dienen. Zudem werden auch Schnittstellen zu externen Softwaresystemen überprüft. Das Risiko solcher Schnittstellen liegt darin, dass nur die >>Hälfte<< einer solchen Schnittstelle nach außen durch das Entwicklerteam kontrolliert werden kann. Es können also unerwartete Änderungen der Schnittstelle auftreten. Ziel der Integrationstest ist es, Schnittstellenfehler also Fehlerzustände im Datenaustausch bzw. in der Kommunikation zwischen Komponenten zu finden (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 52 ff.).

Es werden folgende Typen des Fehlerzustands unterschieden (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 55):

- Es werden keine oder syntaktisch falsche Daten übermittelt
- Übergebene Daten werden unterschiedlich interpretiert
- Daten werden zum falschen oder verspäteten Zeitpunkt (Timing-Problem) oder in zu kurzen Zeitintervallen (Durchsatz-, Kapazität- oder Lastproblem) übergeben

Diese Fehlerwirkungen äußern sich erst in einer Wechselwirkung zwischen zwei Softwarebausteinen. Wenn nicht funktionale Eigenschaften wie z.B. Last oder Performanz als relevant oder risikobehaftet eingestuft sind, können neben funktionalen Integrationstest auch nicht funktionale Tests eine Rolle spielen (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 55).

4.3.4 Systemtest

Nach abgeschlossenem Integrationstest folgt der Systemtest. Um zu überprüfen, ob die spezifizierten Anforderungen vom Produkt erfüllt werden, wird im Systemtest das System als Ganzes betrachtet. Die Testumgebung sollte der späteren Produktivumgebung möglichst nahekommen (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 60).

4.4 Qualitätsmerkmale

Die Testziele eines Projektes definieren sich anhand der Qualitätsmerkmale. Im Testkonzept wird festgelegt, welche Qualitätsmerkmale in welcher Weise im Test nachgewiesen werden (Spillner, Roßner, Winter, & Linz, 2014, S. 18). Die Norm ISO/IEC 25010 gruppiert die Qualitätsmerkmale nach den drei Perspektiven anwendungsbezogene Qualität, interne- und externe Qualitätskriterien. Anhand von Messverfahren können die Untermerkmale definiert und quantifiziert werden. Sie können somit als Testziel vorgegeben und am Testobjekt gemessen werden (Spillner, Roßner, Winter, & Linz, 2014, S. 18).

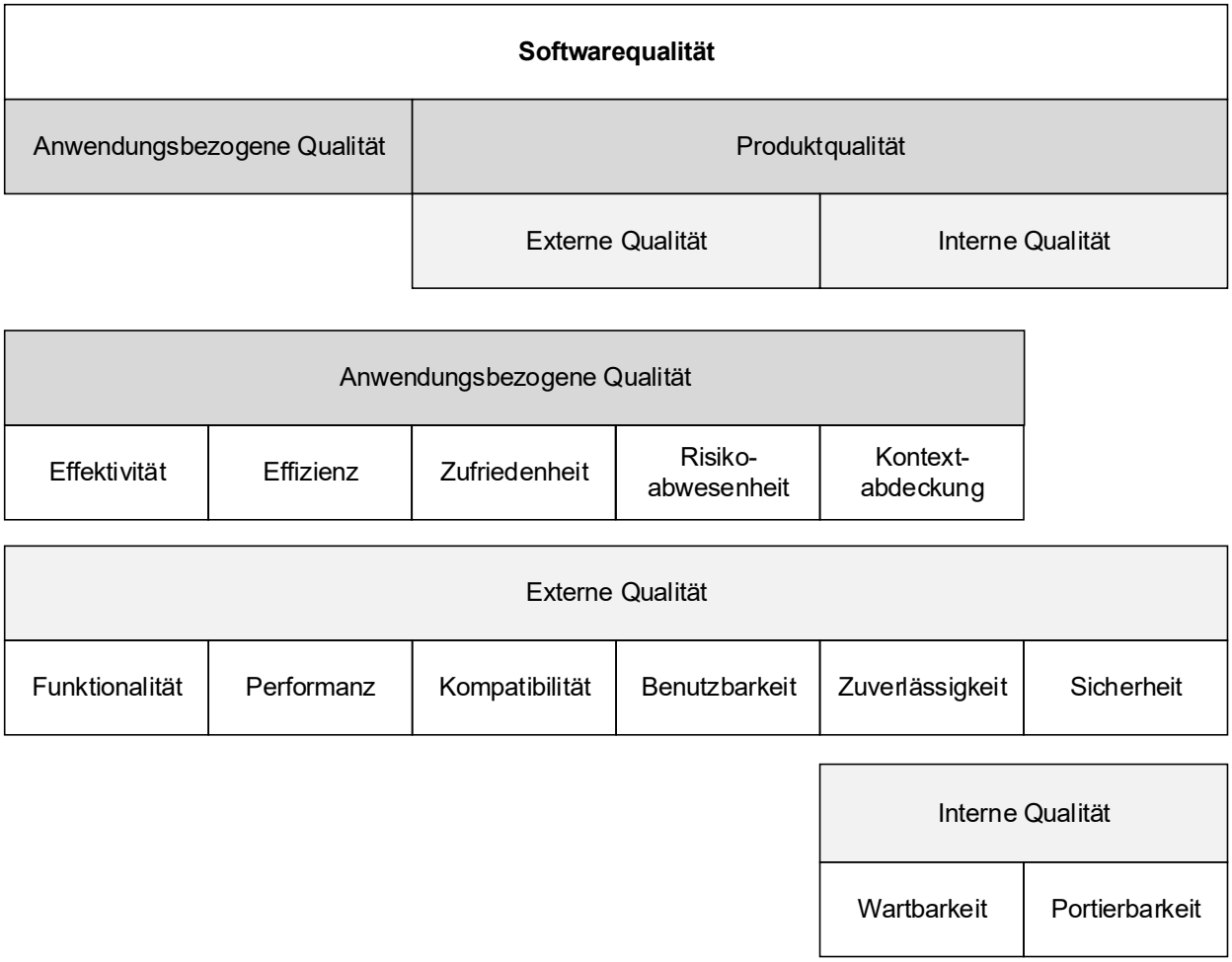

Abb. 4-6 Gruppen der Qualitätsmerkmale von Software nach ISO 25010 (Stauffer, Honegger, & Gisin, 2013, S. 18)

4.5 Testfalldesign

Zur systematischen Erstellung dynamischer Testfälle bieten sich eine Reihe von Verfahren an. Diese Verfahren lassen sich grundsätzlich in zwei Kategorien einteilen. Black- und Whitebox-Testfallentwurfsverfahren. Der Unterschied dieser Verfahren liegt in Ihrem Point of Observation (PoO) und ihrem Point of Control (PoC). Während bei Blackbox-Testfallentwurfsverfahren das Testobjekt als schwarzer Kasten betrachtet wird und das Verhalten des Testobjekts von außen

beobachtet wird (PoO liegt außerhalb des Testobjekts), wird bei den Whitebox Testfallentwurfsverfahren auch auf den Programmtext zurückgegriffen (PoO liegt innerhalb des Testobjekts). Whitebox Testfallentwurfsverfahren werden auch als strukturelle Testfallentwurfsverfahren bezeichnet, da sie die Struktur des Testobjekts berücksichtigen. Bei Blackbox-Testfallentwurfsverfahren steht die Prüfung des spezifizierten Ein-/Ausgabeverhaltens im Mittelpunkt. Deshalb werden diese auch spezifikationsbasierte Testfallentwurfsverfahren genannt. Testen sollte immer eine Kombination von unterschiedlichen Testverfahren umfassen. Die Testverfahren berücksichtigen unterschiedliche Aspekte des Testobjektes und sind entsprechend auch für unterschiedliche Teststufen geeignet. Generell lassen sich Whitebox-Testfallentwurfsverfahren auf die unteren Teststufen wie Komponenten- und Integrationstest anwenden, während sich Blackbox-Verfahren überwiegend für die höheren Teststufen eignen (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 79ff.).

Der Überdeckungsgrad dient als Kriterium für die Testintensität. Dabei dient er vor der Testausführung zur Festlegung, wann die Testintensität als ausreichend angesehen werden soll und nach der Testdurchführung zur Prüfung, ob das geforderte Ziel erreicht wurde. Je intensiver ein Testobjekt getestet werden soll, desto höher ist der zu erreichende Überdeckungsgrad anzusetzen.

4.5.1 Blackbox-Verfahren

ÄQUIVALENZKLASSENBILDUNG

Bei der Äquivalenzklassenbildung wird der Eingabewertebereich in Äquivalenzklassen unterteilt, wobei die Daten innerhalb einer Äquivalenzklasse das gleiche Verhalten des Testobjekts hervorrufen. Es sind Äquivalenzklassen sowohl für gültige als auch ungültige Eingabebereiche zu berücksichtigen. Für das Ableiten von Testfällen wird zunächst der Definitionsbereich je Eingabevariable ermittelt. Diese bilden die Äquivalenzklasse aller zulässigen bzw. erlaubten Eingabewerte. Die Werte außerhalb des Definitionsbereichs werden als Äquivalenzklasse mit unzulässigen Werten betrachtet. Elemente, die laut Spezifikation unterschiedlich verarbeitet werden, sind einer neuen Äquivalenzklasse zuzuordnen. Die Äquivalenzklassen werden solange verfeinert, bis sich alle unterschiedlichen Anforderungen mit den jeweiligen Äquivalenzklassen decken. Für jede Äquivalenzklasse ist ein Repräsentant für einen Testfall auszuwählen. Durch Ergänzung eines vorausgesagten Ergebnisses und ggf. von Vorbedingungen wird der Testfall vervollständigt. Besonders sind die Grenzen der Äquivalenzklassen zu betrachten, da vor allem hier Missverständnisse oder Ungenauigkeiten in den Anforderungen enthalten sind.

Die Systematik trägt dazu bei, dass spezifizierte Bedingungen und Einschränkungen beim Test nicht übersehen werden und keine unnützen Testfälle durchgeführt werden. Der Einsatz der Äqui-

valenzklassenbildung ist bei System- Integrations- und Komponententests möglich. Jedoch werden mögliche Abhängigkeiten und Wechselwirkungen zwischen Bedingungen nicht berücksichtigt. In Kombination mit fehlerorientierten Verfahren, wie der Grenzwertanalyse ist die Äquivalenzklassenbildung ein wirkungsvolles Verfahren (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 114).

```
ÄK-Überdeckung = (Anzahl getestete ÄK / Gesamtzahl ÄK) *100 %
```

GRENZWERTANALYSE

Die Grenzwertanalyse ist eine sinnvolle Ergänzung zu den Testfällen, die durch die Äquivalenzklassen ermittelt wurden, da Fehlerzustände häufig an den Grenzbereichen der Äquivalenzklassen auftreten. Die Methode lässt sich jedoch nur anwenden, wenn die Menge der Daten, die in eine Äquivalenzklasse fallen, geordnet ist und sich die Grenzen identifizieren lassen. Die Ränder der Äquivalenzklassen werden einer Prüfung unterzogen. An jedem Rand wird der exakte Wert und die beiden benachbarten Werte getestet. Für jede Grenze ergeben sich somit drei Testfälle (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 125).

```
GW-Überdeckung = (Anzahl getestete GW / Gesamtzahl GW) * 100%
```

ZUSTANDSBEZOGENER TEST

Bei einer ganzen Reihe von Systemen hat neben den Eingabewerten auch der bisherige Ablauf des Systems Einfluss auf die Berechnung der Ausgaben bzw. auf das Systemverhalten. Die durchlaufene Historie des Systems ist im Test zu berücksichtigen. Zur Veranschaulichung der Historie wird ein Zustandsmodell herangezogen. Beginnend von einem Startzustand können verschiedene Zustände angenommen werden. Zustandsänderungen und –Übergänge werden durch Ereignisse ausgelöst. Bei einem zustandsbezogenen Test kann das Testobjekt ein komplettes System mit unterschiedlichen Systemzuständen, aber auch eine Klasse mit verschiedenen Zuständen sein. Immer wenn die Historie zu unterschiedlichen Verhalten führt, ist ein zustandsbezogener Test durchzuführen (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 133).

Überdeckungskriterien:

- Jeder Zustand wurde mindestens einmal erreicht
- Jeder Zustandsübergang wurde mindestens einmal ausgeführt

- `Alle spezifikationsverletzenden Zustandsübergänge wurden geprüft`

URSACHE- WIRKUNGSGRAPH - ANALYSE

Bisher wurden die Eingabeparameter eines Testobjektes immer unabhängig voneinander betrachtet. Die Ursache-Wirkungs-Graph-Analyse berücksichtigt solche Abhängigkeiten. Voraussetzung ist, dass Ursachen und deren Wirkungen aus der Spezifikation ermittelt werden können. Jede Ursache wird als eine Bedingung beschrieben, die aus Eingabewerten oder Kombinationen von Eingabewerten besteht. Die Eingaben werden über logische Operatoren verknüpft. Die Bedingung und damit die Ursache kann wahr oder falsch sein. Die Wirkungen werden analog behandelt und im Graph vermerkt (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 141).

ANWENDUNGSFALLBASIERTE TESTFÄLLE

Die Grundlage anwendungsfallbasierter Testfälle sind Use-Case Diagramme. Da die Außensicht modelliert ist, ist das Vorgehen für den System- und Akzeptanztest sehr gut geeignet. Wenn mit den Diagrammen Systemkomponenten modelliert werden, können auch Integrationstests abgeleitet werden. Die Diagramme stellen die üblichen oder wahrscheinlichen Abläufe des Systems dar, somit wird die typische Nutzung des Systems geprüft (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 145).

BEWERTUNG BLACKBOX-VERFAHREN

Grundlage der Blackbox-Verfahren sind immer die Anforderungen und die Spezifikation des Systems. Fehler werden nur bei Korrektheit dieser Informationen entdeckt. Zudem wird nicht festgestellt, ob über die Spezifikation hinausgehende Funktionen implementiert wurden. Im Mittelpunkt steht die Prüfung der Funktionalität des Testobjekts.

4.5.2 Whitebox-Verfahren

ANWEISUNGSTEST

Beim Anweisungstest stehen die einzelnen Anweisungen des Testobjektes im Mittelpunkt. Es sind Testfälle zu identifizieren, die eine zuvor festgelegte Mindestquote oder auch alle Anweisungen des Testobjektes zur Ausführung bringen. Hierzu wird zunächst der Anweisungstext in einen Kontrollflussgraphen transferiert. Der Überdeckungsgrad kann dann anhand des Graphen spezifiziert werden. Anweisungen werden als Knoten und Kontrollflüsse als Kanten dargestellt. Nach der Ausführung ist nachzuweisen welche einzelnen Anweisungen ausgeführt wurden. Wenn der vorher festgelegte Überdeckungsgrad erreicht ist, wird der Test als ausreichend angesehen und beendet (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 150).

```
Anweisungsüberdeckung = Anzahl durchlaufene Anweisungen / Gesamtzahl Anwei-
                                    sungen) * 100
```

ZWEIGTEST / ENTSCHEIDUNGSTEST

Ein weiteres Kriterium ist die Überdeckung der Zweige des Kontrollflussgraphen. Hierbei stehen die Kanten des Graphen im Mittelpunkt der Untersuchung. Es wird die Auswertung einer Entscheidung betrachtet. Aufgrund des Ergebnisses wird entschieden, welche Anweisung als nächstes ausgeführt wird. Wenn der Kontrollflussgraph mit seinen Knoten und Kanten Grundlage der Überlegung ist, wird von Zweigtest bzw. Zweigüberdeckung gesprochen. Wenn der Programmtext als Grundlage dient, wird von Entscheidungstests bzw. Entscheidungsüberdeckung gesprochen. Die Zweigüberdeckung ist das umfassendere Kriterium im Verhältnis zur Anweisungsüberdeckung. Allerdings sind auch mehr Testfälle als bei Anweisungstest erforderlich (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 153).

```
ZWEIGÜBERDECKUNG = (ANZAHL DURCHLAUFENE ZWEIGE / GESAMTZAHL ZWEIGE) * 100%
```

BEWERTUNG WHITEBOX-VERFAHREN

Whitebox-Testfallentwurfsverfahren sind besonders für die unteren Teststufen geeignet. Wobei das Konzept der Überdeckung auch auf höheren Teststufen sinnvoll ist. Beim Integrationstest kann so ermittelt werden, welcher prozentuale Anteil von Modulen oder, Komponenten überdeckt wurde. Ein Problem der Whitebox-Verfahren ist, dass nicht vorhandener Programmcode unberücksichtigt bleibt. Anforderungen die übersehen und daher nicht realisiert wurden, werden durch die Whitebox-Verfahren nicht aufgedeckt (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 164).

4.5.3 Intuitive und erfahrungsbasierte Testfallermittlung

Durch die intuitive Testfallermittlung werden systematisch ermittelte Testfälle sinnvoll ergänzt. Erfahrungsbasierte Testfälle können Fehlerwirkungen aufdecken, die durch systematisches Testen übersehen wurden. Deshalb sind sie eine wichtige Ergänzung. Grundlage der Überlegung sind die intuitiven Fähigkeiten und die Erfahrungen, Testfälle nach erwarteten Fehlerzuständen und -wirkungen auszuwählen. Sind die Dokumente, die als Grundlage für die Erstellung der Testfälle heranzuziehen sind, von sehr schlechter Qualität, veraltet oder gar nicht vorhanden, kann das explorative Testen helfen. Dies ist vor allem bei zeitlichen Restriktionen geeignet. Es findet

keine vorherige Planung der Aktivitäten statt. Die Testaktivitäten werden nahezu parallel ausgeführt und folgen keinem strukturierten Prozess. Die möglichen Elemente des Testobjekts werden erforscht. Darauf basierend werden Testfälle ausgewählt. Das Ergebnis wird analysiert. Das unbekannte Verhalten des Testobjektes wird aufgeklärt. Auffälligkeiten und Informationen dienen zur Erstellung weitere Testfälle. Ein Ergebnis des explorativen Tests kann sein, dass klar wird, welche Testverfahren sinnvollerweise einzusetzen sind. Eine Beschränkung auf bestimmte Elemente des zu testenden Programms ist sinnvoll. Die Zergliederung der zu testenden Elemente wird als Test Charter bezeichnet. Der Test eines Charters soll nicht mehr als ein bis zwei Stunden in Anspruch nehmen (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 165).

Besondere Merkmale des explorativen Testens sind Ergebnisse eines Testfalls, die die Erstellung und Ausführung der weiteren Testfälle beeinflussen. So entsteht ein mentales Modell des zu testenden Programms: Gegen dieses Modell wird getestet. Das explorative Testen findet besonders auf höheren Teststufen Anwendung; da auf den niedrigeren Teststufen meist ausreichende Informationen für andere Verfahren zur Verfügung stehen. Exploratives Testen dient vor allem der Vervollständigung von Testfällen und zur Unterstützung der methodischen Verfahren. So können Lücken und Fehler bei der Risikoanalyse aufgezeigt werden. Die Effektivität des explorativen Testens hängt jedoch stark mit den Fähigkeiten und der Intuition der Tester zusammen (Spillner & Linz, Spillner, Linz 2010 – Basiswissen Softwaretest, 2010, S. 166).

4.6 Automatische Testdurchführung

Automatische Tests sind im Regelfall wiederholbar und im Sinne eines Regressionstests angelegt. Die Durchführung sollte im allgemeinen Testprozess abgebildet werden. Die Eingliederung in das gesamte Projekt kann in verschiedenen Formen erfolgen. Für den Einsatz des automatisierten Tests können je nach Situation unterschiedliche Varianten mit unterschiedlichen Schwerpunkten sinnvoll sein (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 40). In inkrementellen Entwicklungsmodellen liegt der Fokus auf der ständigen Testdurchführung. Eine regelmäßige Durchführung von automatisierten Unit Tests kann dem Entwickler unmittelbar Rückmeldung über die Auswirkungen seiner Änderungen geben. Diese Art der Testdurchführung repräsentiert im Kern einen Regressionstest, der sicherstellen soll, dass die Änderungen bereits korrekt funktionierende Programmteile nicht beeinflussen. Nimmt die Durchführung der Gesamtmenge an Unit Tests zu viel Zeit in Anspruch, ist es wichtig entweder ein repräsentatives, gleichbleibendes Set an Testfällen aus der Menge der verfügbaren Unit Test, oder anhand der Codeänderungen ein Set von Testfällen, auszuwählen. In beiden Fällen sollte jedoch das vollständige Set an Unit Test regelmäßig durchgeführt werden. Tests, deren Durchführung nach jeder Änderung am zu testendem System zu lange dauern würde, können alternativ in relativ kurzen, aber mehrtägigen

Abständen durchgeführt werden. Im Allgemeinen ist dieses Zeitfenster ausreichend, um eine vollständige Durchführung von automatisierten Unit Tests unterbringen zu können. Automatisierte Integrations- bzw. Systemtests sind aufgrund ihrer vergleichsweise langen Laufzeit nicht für diese Varianten der Durchführung geeignet. Hier empfiehlt sich, zumindest einmal pro zu lieferndem Inkrement, die vollständige Menge an automatisierten Testfällen laufen zu lassen. Ist dies nicht möglich bietet sich die Risikoanalyse für die Auswahl durchzuführender Testfälle an (Bucsics, Baumgartner, Seidl, & Gwihs, 2015, S. 43).

4.7 Risikoorientierung

Für die Auswahl der tatsächlich zu realisierenden und durchzuführenden Tests bieten sich risikoorientierte Ansätze an. Intention des risikoorientierten Testens ist es, Risiken durch gezielte Tests zu minimieren. Das risikoorientiere Testen nutzt das Wissen über die Risiken dazu, den Inhalt des Testkonzeptes so festzulegen, dass die Risiken unter den gegebenen Rahmenbedingungen möglichst minimiert werden. Darüber hinaus kann Testen auch über die Risikobewertung selbst informieren. Anhand der wichtigsten Risiken sind gemäß einer risikoorientierten Teststrategie geeignete Testverfahren auszuwählen. Zur Einbeziehung der Risiken im Rahmen des Testmanagements gehören folgende Aktivitäten, die iterativ während des gesamten Projekts durchgeführt werden:

- Risikoidentifizierung
- Risikobewertung
- Risikobeherrschung
- Risikomanagement

Risikomanagementaktivitäten sind zum frühestmöglichen Projektzeitpunkt zu beginnen und laufend fortzuführen (Spillner, Roßner, Winter, & Linz, 2014, S. 94). Das Wissen um die Risiken wird dazu genutzt, das Testkonzept und den Testplan festzulegen. Hierzu werden geeignete Testentwurfsverfahren gemäß den Risikokategorien bestimmt, sowie Tests entsprechend priorisiert (Spillner, Roßner, Winter, & Linz, 2014, S. 115).

4.8 Metriken

Testmetriken erlauben quantitative Aussagen bezüglich der Produktqualität. „Sie müssen auf solider maßtheoretischer Grundlage definiert und validiert werden“ (Spillner, Roßner, Winter, & Linz, 2014, S. 191) Anhand des Messobjektes lassen sich testfallbasierte Metriken, testbasis- und testobjektbasierte Metriken und Kosten bzw. aufwandsbasierte Metriken differenzieren (Spillner,

Roßner, Winter, & Linz, 2014, S. 191). Messen ist somit eine zentrale Aufgabe des Testmanagements. In der Softwaretechnik und im Testumfeld meint der Begriff „Metrik“ sowohl die Messverfahren als auch die Größe selbst, mit denen bestimmte Eigenschaften am Testobjekt gemessen werden (Spillner, Roßner, Winter, & Linz, 2014, S. 193). IEEE definiert eine Softwarequalitätsmetrik als „[…] eine Funktion, die bestimmte Eigenschaften von Softwareentwicklungsprozessen oder dem erstellten Zwischen- oder Endprodukt auf Werte eines vorgegebenen Wertebereichs abbildet.“ Diese Werte dienen als Erfüllungsgrad bestimmter Eigenschaften. Die Messung bildet bestimmte Eigenschaften, der zu vermessenden Objekte eines sogenannten empirischen Relationssystems mittels eines Messverfahrens auf bestimmte Werte eines formalen Relationssystems ab (Spillner, Roßner, Winter, & Linz, 2014, S. 194).

4.9 Normen und Standards

Normen und Standards fördern die Einheitlichkeit von Produkten und Prozessen und definieren allgemein anerkannte Regeln der Technik und damit den fachlichen – rechtliche Bezugsrahmen, innerhalb dessen sich Projekte bewegen (Spillner, Roßner, Winter, & Linz, 2014, S. 277). Diese Regeln sind oft in nationalen und internationalen Normen- und Industriestandards beschrieben. Auf internationaler Ebene sind die „International Organization for Standardization“ (ISO) und die „International Electrotechnical Commission“ (IEC) verantwortlich. Branchenspezifische Organisationen kooperieren mit Standardisierungsorganisationen. Im Bereich Softwaretests definiert das „International Software Testing Qualifications Board“ (ISTQB) und seine nationalen Untergliederungen die Standardisierung (Spillner, Roßner, Winter, & Linz, 2014, S. 278). Normen und Standards lassen sich nach Kategorien klassifizieren, so zum Beispiel nach ihrer normativen Zielrichtung, nach ihrer Anwendbarkeit im Softwarelebenszyklus oder der Granularität des Anwendungsgebietes (Spillner, Roßner, Winter, & Linz, 2014, S. 285).

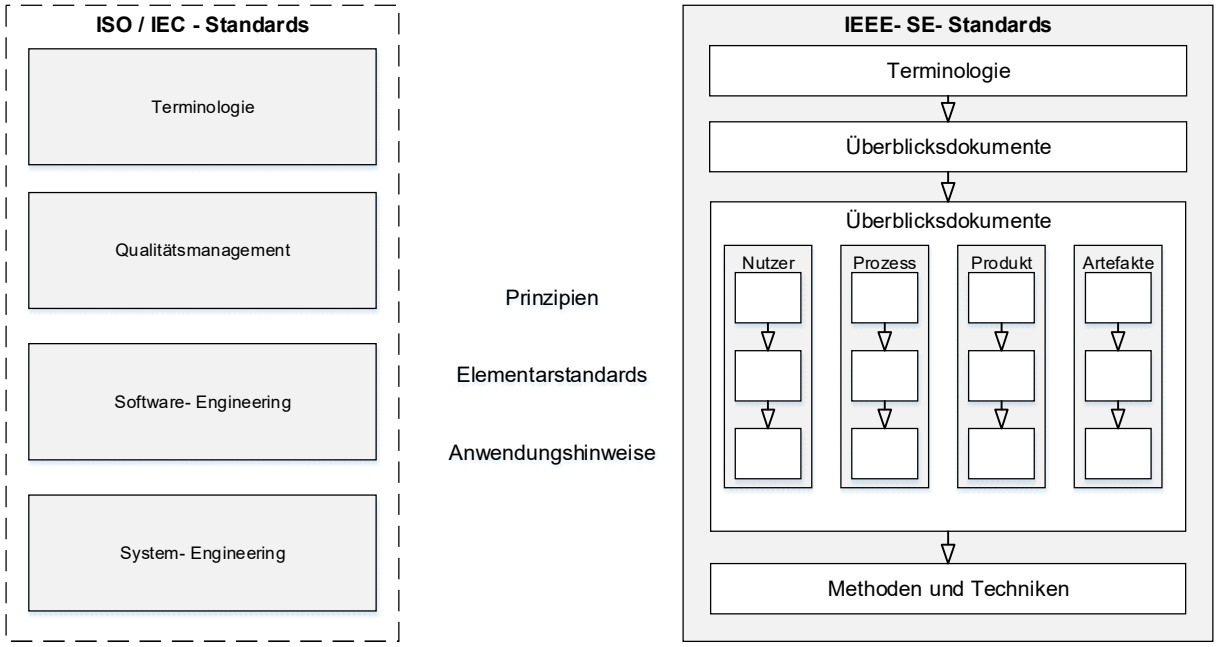

Abb. 4-7 Gliederung der ISO/IEC- und IEEE-SE-Standards (Spillner, Roßner, Winter, & Linz, 2014, S. 286)

Die obigen Kategorien lassen sich pragmatisch vereinfachen (Spillner, Roßner, Winter, & Linz, 2014, S. 287).

- Terminologie- und Vertragsnormen
- Prozessnormen
- Produkt- und Dokumentationsnormen
- Methoden- und Techniknormen

Die Inhalte und Aussagen der Normen lassen sich nur unter Verwendung festgelegter Begriffe einheitlich, eindeutig und konsistent niederschreiben. Deshalb bilden Terminologie- und Vertragsnormen, die die Begriffe eines Fachgebietes definieren, die Basis für alle anderen Normen (Spillner, Roßner, Winter, & Linz, 2014, S. 287). Unter Prozessnormen werden alle branchenübergreifenden Standards subsummiert, die Mindestanforderungen an Prozesse bzw. deren Bewertung und Verbesserung spezifizieren (Spillner, Roßner, Winter, & Linz, 2014, S. 288). Dokumente- und Produktvorgaben sowie Qualitätsmerkmale inkl. geeigneter Mittel zu ihrer Feststellung fallen unter die Rubrik Produkt- und Dokumentationsnormen. In Bezug auf die Qualitätsziele finden sich in der für das Testmanagement wichtigen Normenreihe ISO/IEC-255xx „Software Engineering – Software Produkt Quality Requirements and Evaluation (SQuaRE)“ ein Qualitätsmodell sowie konkrete Vorgaben inklusiver Metriken zur Bewertung der Gebrauchsqualität. Sie unterstützen den Testmanager bei der Testplanung hinsichtlich der Festlegung von Testzielen sowie von Mess- bzw. Testverfahren für einzelne Qualitätsmerkmale (Spillner, Roßner, Winter, & Linz, 2014, S. 291).

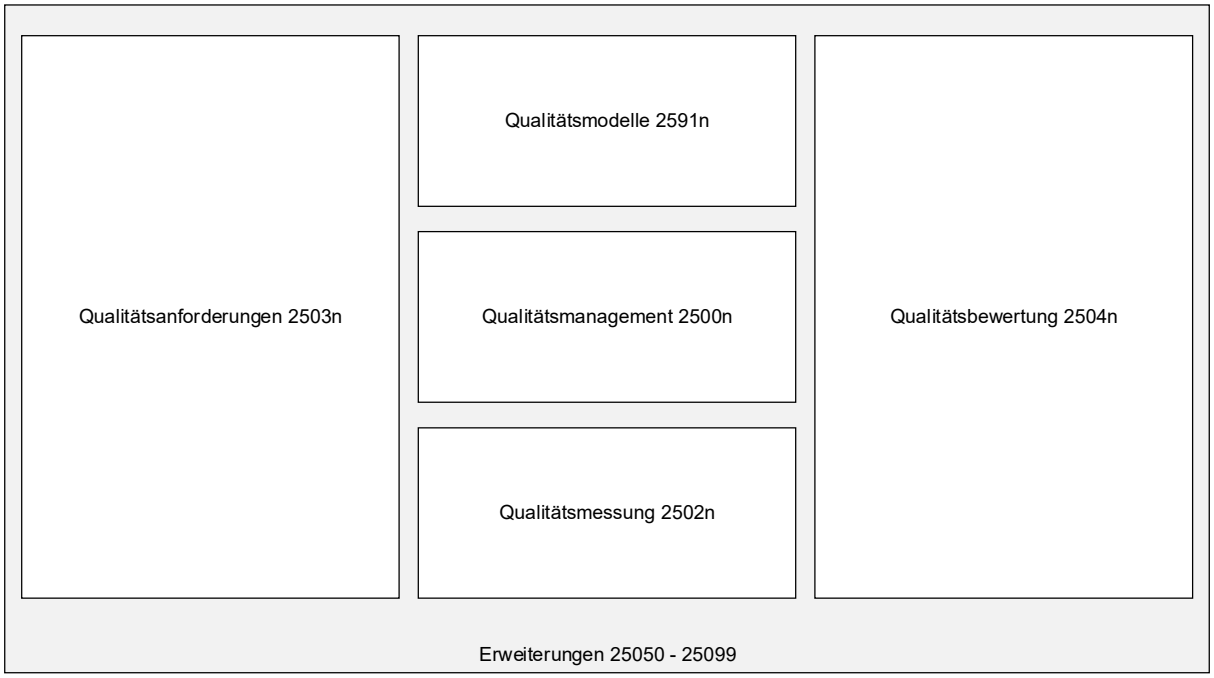

Abb. 4-8 Bereiche der Normen-Reihe ISO/IEC 250x (Spillner, Roßner, Winter, & Linz, 2014, S. 292)

Das Testkonzept und die Testdokumentation werden in den Standards IEEE 739 „IEEE Standard for software Quality Assurance Plans“ für das Testkonzept und IEEE 829 „IEEE Standard for Software Test Documentation“ für die gesamte Testdokumentation behandelt (Spillner, Roßner, Winter, & Linz, 2014, S. 292). Konkrete Verfahrensanweisungen für die konstruktiven, prüfenden und unterstützenden Tätigkeiten der Softwareentwicklung werden in den Methoden- und Techniknormen abgedeckt. Für das Testen relevant ist hier besonders die Standardreihe ISO/IEC/IEEE 29119 „Software and Systems Engineering – Software Testing“. Die Normenreihe unterstützt das Testen im kompletten Softwarelebenszyklus, vom statischen Test der Anforderungen über Komponenten-, Integrations-, System- und Benutzerakzeptanztests bis hin zum Testen während des Produkteinsatzes (Spillner, Roßner, Winter, & Linz, 2014, S. 294).

4.10 Testdokumentation

„Eine gute Dokumentationsarchitektur hat das Ziel, die Wiederverwendung von Dokumentationsbestandteilen und die Vereinheitlichung der Dokumentationsarbeit zu unterstützen.“ (Spillner, Roßner, Winter, & Linz, 2014, S. 163).

Die Dokumentationsarchitektur definiert die Wiederverwendung von Dokumentationsbestandteilen und die Vereinheitlichung von Dokumentationsarbeiten. Hierzu werden Inhalte auf verschiedenen Organisationsebenen angesiedelt, die auf den darauf aufbauenden Ebenen einheitlich bleiben (Spillner, Roßner, Winter, & Linz, 2014, S. 165).

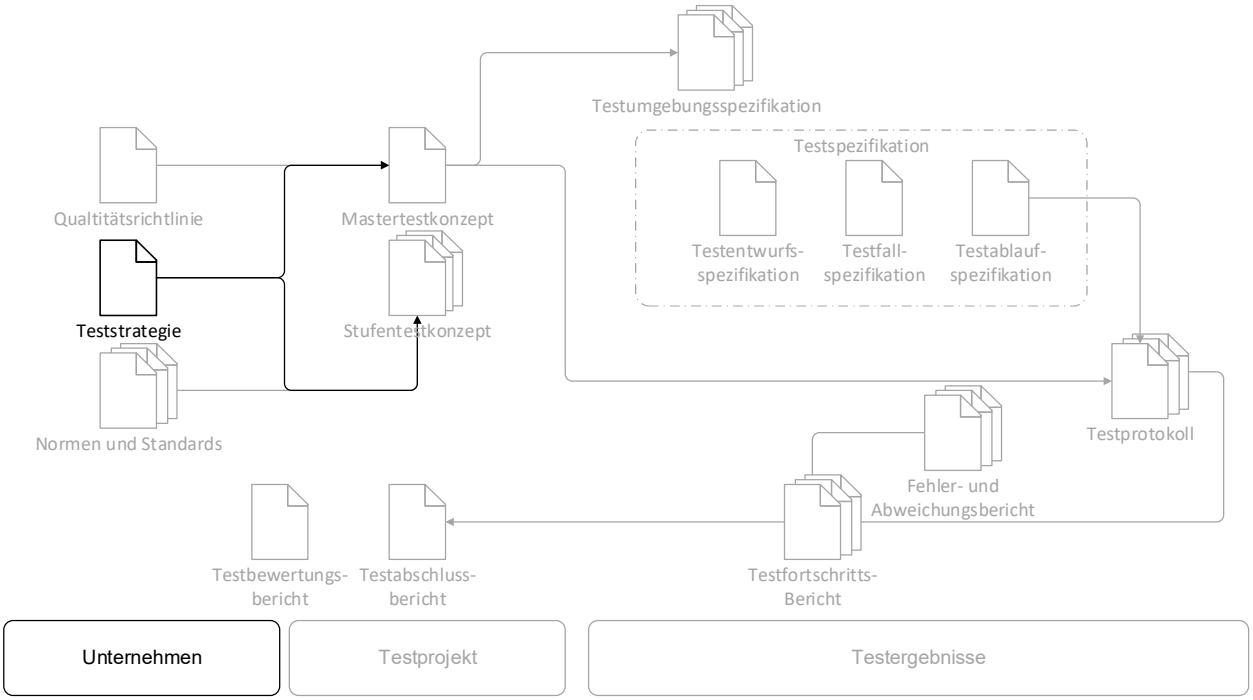

Abb. 4-9 Einordnung der Teststrategie in die Dokumentationsarchitektur (Winter, Roßner, Brandes, & Goetz, 2016, S. 293)

Dokumente auf Organisationsebene

In der Qualitätsrichtlinie werden die Qualitätsziele des Managements beschrieben und die Aufgaben sowie die strategische Bedeutung von Qualitätssicherung und Qualitätsmanagement zur Erreichung dieser Ziele festgelegt. Die Testorganisation konkretisiert die allgemeinen Aussagen der Qualitätsrichtlinie in einer Testrichtlinie. Aufbauend auf den allgemeinen Dokumenten werden eine oder mehrere Teststrategien erstellt. Die Teststrategie ist der rote Faden von den Anforderungen an das zu testende System hin zu einzelnen Testaktivitäten (Spillner, Roßner, Winter, & Linz, 2014, S. 165).

Dokumente auf Projektebene

Für jedes einzelne Entwicklungsprojekt wird aus der Teststrategie ein Mastertestkonzept abgeleitet, dass festlegt welche Teststufen in einem Projekt erforderlich sind. Für diese beschreibt das Stufentestkonzept die Auslegung der Teststrategie (Spillner, Roßner, Winter, & Linz, 2014, S. 165).

Dokumente auf Projektebene

Für jedes einzelne Entwicklungsprojekt wird aus der Teststrategie ein Mastertestkonzept abgeleitet, das festlegt welche Teststufen in einem Projekt erforderlich sind. Für dieses beschreibt das Stufentestkonzept die Auslegung der Teststrategie (Spillner, Roßner, Winter, & Linz, 2014, S. 165).

TESTSTRATEGIE / TESTHANDBUCH

Eine Teststrategie kann langfristige unternehmensweite Gültigkeit haben oder gemeinsame Regelungen für mehrere ähnliche Projekte beinhalten. Die hier gewählte Dokumentationsarchitektur sieht ein Testhandbuch vor, das in möglichst vielen Projekten verwendbar ist und als gemeinsame Basis für projektspezifische Testkonzepte dient. Zur Planung der Tests in einem neuen Projekt soll das Testhandbuch als Auswahlkatalog für die konkrete Projektumsetzung herangezogen werden können. Das Testkonzept muss dann lediglich projektbezogene Konkretisierungen beschreiben. Das Testhandbuch muss somit Antworten auf Fragen bieten, die zu Beginn eines jeden Projekts hinsichtlich der zu planenden Tests immer wieder gestellt werden (Spillner, Roßner, Winter, & Linz, 2014, S. 170).

DEFINITION DER TESTSTRATEGIE

Die Teststrategie bildet den roten Faden von den Anforderungen an das zu testende System hin zu den einzelnen Testaktivitäten. Die zur Prüfung der Testobjekte einzusetzenden Testverfahren und die Priorisierung der zu testenden Systemteile und Qualitätsmerkmale werden hier festgelegt. Die Teststrategien unterscheiden sich je nach Organisations- und Projektkontext. Es wird zwischen analytischen Teststrategien wie z.B. das risikoorientierte Testen oder das modellbasierte Testen und reaktiven Strategien unterschieden. Die Teststrategie kann allgemein und unabhängig vom konkreten Testprojekt aber auch projektspezifisch formuliert werden. Sind beide Varianten im Unternehmen vorhanden, sollten möglichst viele Teile der Teststrategie übergreifend im Testhandbuch formuliert werden (Spillner, Roßner, Winter, & Linz, 2014, S. 15). Gemäß dem Risiko sind bei der Bestimmung der Teststrategie unterschiedliche Testentwurfsverfahren und Endkriterien festzulegen. Wobei kritische Systemteile intensiv getestet werden müssen.

5 BI TESTEN

Eine Teststrategie für die iterative BI-Entwicklung muss Anforderungen aus den Bereichen Organisation, BI-Architektur und Testmethoden erfüllen. In diesem Kapitel werden verschiedene Ansätze für das Testen von BI-System vorgestellt. Eine Einordnung erfolgt anhand der Anforderungen, die sich aus den drei Teilbereichen Organisation, BI-Architektur und Testmethoden ergeben.

Kriterium	**Beschreibung**
Organisation	
Prozessdefinition	Die Strategie gibt einen definierten Prozess vor.
Rollendefinition	Aus dem Prozess ergibt sich eine Rollendefinition.
Agil Iterativ	Das Vorgehen unterstützt die BI Agilität und iterative Entwicklung.
Teststufen	Es werden für die Teststufen spezifische Testfälle genannt.
BI – Architektur	Die Teststrategie unterstützt die Besonderheiten der einzelnen Architekturebenen.
Methoden	
Testfallentwurf	Die genannten Testfallentwurfsmethoden leiten sich aus den Qualitätsmerkmalen eines BI-Systems ab.
Test Priorisierung	Es werden geeignete Methoden genannt, Testfälle zu priorisieren.
Metriken	Die Messbarkeit der Softwarequalität wird durch geeignete Metriken unterstützt.
Automatisierung	Es werden Ansätze für eine Automatisierung von Testfällen beschrieben.

Tabelle 5.1 Bewertungskriterien für eine BI Teststrategie

5.1 Stauffer, Honegger, & Gisin, 2013

Stauffer, Honegger, & Gisin, sind als Dozenten für Business Intelligence und Data Warehousing tätig. Arbeitsschwerpunkte sind hierbei Systemarchitektur und Multidimensionale Datenmodelle, sowie qualitative Themen wie Datenqualität und Testen. Sie beschreiben in Ihrem Buch ein umfassendes Konzept zum Testen von BI- Systemen. Andere Autoren (Trahasch & Zimmer, 2016), (Krawatzeck, Zimmer, & Trahasch, 2013) beziehen sich in Ihren Aussagen auf die in (Stauffer, Honegger, & Gisin, 2013) dargelegten Erkenntnisse.

5.1.1 Methode und logische Testfälle

Stauffer, Honegger, & Gisin stellen ein Referenzmodell für das Testen vor, um die Aspekte des Testens möglichst umfassend zu behandeln. Das Modell setzt auf bestehende Standardsund wurde punktuell um eigene Vorgehensweisen ergänzt. Der Testprozess wird aus drei Perspektiven betrachte:

- Testphasen
- Organisationsebenen
- Elemente

Die Phasen entsprechen denen des allgemeinen Testprozesses. Organisatorisch teilen sie den Prozess auf die Ebenen Testmanagement und Testoperation auf. Die Ebene Testmanagement wird durch das gewählte Vorgehensmodell bestimmt. Die Ebene Testoperation wird durch die zu prüfenden Qualitätskriterien und Elemente geprägt. Vorallem diese Ebene zeichnet sich durch die Spezialitäten von Business Intelligence aus. [S.18] Bei den Elementen unterscheiden Stauffer, Honegger, & Gisin die Aktivitäten, denen Menschen und Rollen sowie Werkzeuge zugeordnet sind. Anhand der drei Perspektiven beschreiben sie den Testprozess in den Phasen Planung, Testdurchführung und Abschluss.

TESTANALYSE & TESTDESIGN

Um geeignete Testfälle abzuleiten und qualitative Schwerpunkte zu setzen, beschreiben Stauffer, Honegger, & Gisin ein Vorgehen in fünf Schritten, wobei die Definition der Testfälle ausschließlich in der Planungsphase des Testprozesses angesiedelt wird (Stauffer, Honegger, & Gisin, 2013, S. 61):

- Schritt 1: Aufzeichnen des gesamten Systems
- Schritt 2: Kennzeichen der zu testenden Komponenten
- Schritt 3: Ableiten von Testfällen
- Schritt 4: Definition der zu überprüfenden Qualitätskriterien, z.B. auf Basis von ISO 9126
- Schritt 5: Ableiten der geeigneten Testverfahren aus den Kategorien statische oder dynamische Tests

Der erste Schritt umfasst die grafische Darstellung des gesamten Systems mit all seinen Komponenten, wobei auf Form und Farbe verzichtet wird. Hervorgehoben wird die explizite Darstellung von Schnittstellen und Verbindungen. Einflussrichtungen werden durch das einzeichnen von Pfeilen visualisiert. Als Testbasis dienen Flussdiagramme, Pläne der Architektur oder auch das Datenmodell (Stauffer, Honegger, & Gisin, 2013, S. 61).

Anschließend werden in einem zweiten Schritt die Komponenten des Systems gekennzeichnet. Beispielhaft werden folgende Farbcodes und Symbole angebracht (Stauffer, Honegger, & Gisin, 2013, S. 63):

- Rot durchstreichen für zu entfernende Komponenten
- Orange für alles, was neu ist
- Grün für zu ändernde Komponenten
- Gelb für alles, was sich zwar nicht ändert, ab durch die Änderungen einer vorherigen Komponente betroffen ist. Hier spielt die Einflussrichtung eine wichtige Rolle (Pfeile)

Anhand der Kennzeichnung werden im dritten Schritt Testfälle abgeleitet. Der Fokus liegt auf neuen und sich ändernden Komponenten. Zudem werden betroffene Komponenten mithilfe von Regressionstests geprüft. Komponenten, die sich nicht ändern oder nicht von den Änderungen betroffen sind, werden nicht getestet (Stauffer, Honegger, & Gisin, 2013, S. 64).

LOGISCHE TESTFÄLLE ANHAND DER ARCHITEKTUR

Logische Testfälle werden anhand des Architekturmodells unter der Annahme, das ähnliche Systemkomponenten, ähnliche Testfälle zur Folge haben, gruppiert. Daraus ergibt sich folgende Klassifizierung (Stauffer, Honegger, & Gisin, 2013, S. 182):

- Datenintegration und Schnittstellen zu den Datenquellen
- Datenspeicherung
- Informationsbereitstellung
- Plattform und Infrastruktur
- Datengetriebene Testfälle

Datenintegration und Schnittstellen zu den Datenquellen

Da keine Tests der Quellsysteme vorgesehen sind, werden nur die Schnittstellen in Form von Ladeprozessen überprüft. Hierbei wird zwischen initialen Ladevorgängen und den darauffolgenden Ladeprozessen unterschieden (Stauffer, Honegger, & Gisin, 2013, S. 185):

- Initialer Ladevorgang
- Korrekte Verarbeitung eines zweiten Ladevorgangs (update/insert)
- Erkennen und Verhindern eines doppelten Ladevorgangs
- Korrektes Abarbeiten von mehreren Dateien
- Rollback bei fehlerhaftem Laden
- Zurückweisen von fehlerhaften Daten
- Durchlaufzeit der Ladevorgänge

Datenspeicherung

Zu prüfen ist, ob das Datenmodell alle erwarteten Felder enthält und ob alle Attribute erwartete Werte enthalten. Ein Vergleich der Anzahl der Datensätze in Eingabedatei und Zielschema und das Bilden von Quersummen und Prüfsummen über Werte, können wie das Zählen von Dimensionsattributen automatisiert werden. Prüfungen können kombiniert werden. Staufer et al. unterscheiden zwei Standard Testfälle (Stauffer, Honegger, & Gisin, 2013, S. 192):

- Alle Fakten sind existierenden Dimensionseinträgen zugeordnet
- Vollständig ausbalancierte Hierarchien

Informationsbereitstellung

Für das Testen des Frontendes werden Checklisten verwendet. Die Prüfungen umfassen die Namen, Konventionen, das Layout und das funktionale Verhalten wie zum Beispiel das Filtern.

5.1.2 Daten

Die datengetriebenen Testfälle werden nach den Testschwerpunkten Datenqualität und Softwarequalität geordnet. Wobei jeweils zwischen Validierung, also der Korrektheit und Verifizierung also der Überprüfung auf Vollständigkeit der Daten, unterschieden wird (Stauffer, Honegger, & Gisin, 2013, S. 87 f.).

Testschwerpunkt	**Validieren**	**Verifizieren**

Softwarequalität	**Tun wir die Dinge richtig?**	**Tun wir die richtigen Dinge?**
Funktional	Werden unvollständige Quelldaten erkannt? Wie reagiert das System bei Veränderungen im Quellsystem?	
Nicht funktional	▪ Interpretierbarkeit ▪ Nachvollziehbarkeit ▪ Sicherheit ▪ Antwort- und Laufzeiten	

Tabelle 5.1-1: Testschwerpunkte nach Datenqualitätsmerkmal (Stauffer, Honegger, & Gisin, 2013, S. 87)

FUNKTIONALE SOFTWAREQUALITÄT IN BEZUG ZU DEN DATEN

Die funktionale Softwarequalität in Bezug zu den Daten stellt die Frage, wie die Prüfung erfolgt, ob Quelldaten vollständig übertragen wurden, und wie erkannt wird, wenn Änderungen im Quellsystem vorgenommen wurden (Stauffer, Honegger, & Gisin, 2013, S. 93).

NICHT FUNKTIONALE SOFTWAREQUALITÄT IN BEZUG ZU DEN DATEN

- Datenvolumen und Laufzeit
  Performancetests beinhalten nicht nur große Datenmengen, sondern berücksichtigen auch Dateninhalte, die durch andere Verarbeitungsschritte zu längeren Laufzeiten führen können.
- Berechtigungen auf Dateninhalte
  Testen der Berechtigungseinstellungen für den Zugriff auf Dateninhalte.
- Starke Schwankungen im Datenvolumen
  Schwankungen in der Datenmenge des Quellsystems müssen verstanden werden. Testfälle müssen nicht nur die Laufzeit (Performance), sondern auch die überlappenden Zeitfenster für Ladeprozesse (Robustheit) prüfen.
- Interpretierbarkeit
- Nachvollziehbarkeit der Daten und Analysen
  Wird das Ergebnis der Datenaufbereitung angezweifelt, so muss jeder einzelne Schritt entlang der Architektur bis zurück in die Quellsysteme aufgezeigt werden. Die Nachvollziehbarkeit der Analysen behandelt Daten vom Zielsystem zurück zu den Quellen. Alle anderen beschriebenen datengetriebenen Testfälle betrachten die Daten aus Sicht der Quellen.

Die datengetriebenen Testfälle können jeweils einem oder mehreren Qualitätskriterien zugeordnet werden. Tabelle 5.2: Zuordnung von Testfällen zu Qualitätskriterien Tabelle 5.2 zeigt diese Zuordnung.

	Funktionalität	Sicherheit	Benutzbarkeit	Wartbarkeit	Robustheit	Effizienz
Fehlende Quelldaten	X				X	
Unvollständige Quelldaten	X				X	
Alte oder veraltete Quelldaten	X				X	
Veränderungen im Quellsystem	X				X	
Syntaktische Richtigkeit	X					
Semantische Fehler in den Quelldaten	X					
Datenvolumen und Laufzeit					X	X
Berechtigungen auf Dateninhalte (Rowlevel Security)		X				
Starke Schwankungen im Datenvolumen à unterschiedliche Laufzeitverhalten					X	X
Interpretierbarkeit der Daten (Daten-verständnis)			X			
Nachvollziehbarkeit der Daten und Analysen	X					

Tabelle 5.2: Zuordnung von Testfällen zu Qualitätskriterien (Stauffer, Honegger, & Gisin, 2013, S. 94)

5.1.3 Testautomatisierung

Stauffer, Honegger, & Gisin sehen für die Verwendung von Testautomatisierungswerkzeugen nur ein eingeschränktes Einsatzgebiet. Anhand der Qualitätskriterien schränken sie geeignete Testszenarien auf die Merkmale Funktionalität, Zuverlässigkeit und Effizienz ein.

5.1.4 **Bewertung**

Staufer, Honnegger & Gisin stellen einen Prozess für das Testen von BI-Systemen vor. Das Vorgehen in der Analyse- und Designphase Testfälle anhand der Architektur auszuwählen, ist ein pragmatischer Ansatz der Risikoorientierung. Die Teststufen entsprechen denen des V-Modells, eine Verteilung der Testfälle auf die Teststufen findet jedoch nicht statt. Der Inhalt der Teststufen wird nicht näher betrachtet. Anhand der Testfallvorlagen lassen sich Anforderungen an Testdaten und wiederverwendbare Testfälle, die durch Metriken ergänzt werden können, ableiten. Wird diese durch ein Risikoinventar ergänzt, kann dies als eine leichtgewichtige Option für einen agilen Entwicklungsprozess gesehen werden.

	Bewertung
Organisation	
Prozessdefinition	[++] Es wird ein strukturierter Prozess definiert, der es ermöglicht, die Testaufgaben für ein Entwicklungsprojekt abzuleiten.
Rollendefinition	[++] Der Prozess gibt durch die Beschreibung von der Organisationsebene einen Anhaltspunkt für das Zuordnen von Rollen.
Agil Iterativ	[-]
Teststufen	[+] Eine Zuordnung von Testfällen zu Teststufen fehlt.
BI – Architektur	[++]
Methoden	
Testfallentwurf	[+] Es findet eine detaillierte Beschreibung von möglichen Testfällen und eine Zuordnung zu Qualitätskriterien statt.
Test Priorisierung	[+] Es werden geeignete Methoden genannt, um Testfälle zu priorisieren.
Metriken	[ ]
Automatisierung	[-] Die Möglichkeit der automatisierten Durchführung von Regressionstests wird genannt. Jedoch wird der Testautomatisierung keine Priorität beigemessen.
Legende: [ ] nicht Beschrieben [-] nicht geeignet [+] wird berücksichtigt [++] wird vollumfassend berücksichtigt	

Tabelle 5.3 Bewertung der BI Teststrategie von (Stauffer, Honegger, & Gisin, 2013)

5.2 Trahasch & Zimmer, 2016

5.2.1 Methode und logische Testfälle

Trahasch & Zimmer übertragen den fundamentalen Testprozess auf den Bereich agile BI. Hierbei erweitert er den Ansatz von (Stauffer, Honegger, & Gisin, 2013) um die Qualitätsmerkmale Kompatibilität und Wartbarkeit und um die Testpunkte *„Schnittstellen zu den Datenquellen“* und *„Metadatenbanksystem*“. Die Priorität sehen sie beim Testen des zentralen Datenbestandes. Das Auftreten möglicher Fehler ist aufgrund der komplexen, zum Befüllen des DWH notwendigen, ETL-Prozesse sehr wahrscheinlich. Des Weiteren können fehlerhafte Daten im DWH zu falschen

Auswertungsergebnissen führen und somit einen hohen Schaden verursachen. Für die Ermittlung möglicher Testarten nennen Trahasch & Zimmer verschiedene Dimensionen. Die Dimensionen bilden einen BI-Testing Cube der 192 Handlungsfelder aufspannt. Pro Handlungsfeld können eigene Testfälle und Testdaten erstellt werden. Wobei die Teststufen denen des V-Modells entsprechen.

- Teststufe
- Systemqualitätsmerkmal
- BI-Testobjekt

TESTZIELE

Trahasch & Zimmer nehmen anhand der zu testenden Qualitätsmerkmale eine Einteilung der Tests vor. Diese Einteilung nach Testziel nehmen sie anhand der im Standard ISO/IEC 25010 (ISO/IEC, ISO 9000:2015, 2015) definierten acht Qualitätsmerkmalen vor. Zusätzlich ergänzen sie die Datenqualität als neuntes Qualitätsmerkmal (Trahasch & Zimmer, 2016, S. 119).

TESTZIELE

Trahasch & Zimmer nehmen anhand der zu testenden Qualitätsmerkmale eine Einteilung der Tests vor. Diese Einteilung nach Testziel nehmen sie anhand der im Standard ISO/IEC 25010 (ISO/IEC, ISO 9000:2015, 2015) definierten acht Qualitätsmerkmalen vor. Zusätzlich ergänzen sie die Datenqualität als neuntes Qualitätsmerkmal (Trahasch & Zimmer, 2016, S. 119).

TESTOBJEKTE ANHAND DER ARCHITEKTUR

Die BI-Architektur teilen sie in ihre Subsysteme auf, um handhabbare Teile zu erhalten und jedes dieser Subsysteme auf allen vier Teststufen zu testen. So soll unter anderem sichergestellt werden, dass die Komponenten für sich korrekt funktionieren und entlang der Datenflüsse korrekt miteinander interagieren. Da B-Systeme von der Korrektheit der Schnittstellen zu den Vorsystemen abhängen, sind dieses ebenfalls zu testen. Hieraus ergeben sich sechs Testobjekte:

- Schnittstellen zu den Datenquellen
- ETL (Datenintegration)
- DWH (Datenhaltung)
- DM (Data Mart, Datenanalyse)
- Frontend (Informationsrepräsentation)
- MD (Metadatenbanksystem)

Die Unabhängigkeit der drei Dimensionen Teststufe, Systemqualitätsmerkmal und BI-Testobjekt führt zum „*BI Testing Cube*“ (Trahasch & Zimmer, 2016, S. 121).

5.2.2 Testautomatisierung

Im Hinblick auf die Teststufen und die BI-Testobjekte können alle Tests automatisiert werden. Jedoch finden sich Einschränkungen hinsichtlich der Qualitätsmerkmale. Einige Qualitätsmerkmale lassen sich ausschließlich durch Automatisierung testen andere schließen eine Automatisierung aus. Wobei die Teilmerkmale funktionale Angemessenheit sowie das Merkmal Gebrauchstauglichkeit sich nicht automatisch überprüfen lassen. Im Kontext von BI-Projekten sind vor allem die Qualitätsmerkmale funktionale Tauglichkeit, Zuverlässigkeit und Performanz für automatisierte Tests geeignet. „Die Datenqualität stellt erneut eine Besonderheit dar. Da diese nicht von der Ausführung des BI-Systems abhängt, sondern die Bewertung eines gegebenen Zustands repräsentiert, ist sie lediglich durch statische Tests zu überprüfen. Folglich lassen sich nur jene Datenqualitätsteilmerkmale automatisiert testen, die sich formalisieren lassen (Trahasch & Zimmer, 2016, S. 125).

5.2.3 Bewertung

Eine Aufteilung der BI-Architektur in ihre Subsysteme ist mit Hinblick auf die Handhabbarkeit der Testobjekte sinnvoll. So lassen sich einzelne Teilsysteme zunächst als Komponente testen und anschließend zu einem gesamten System integrieren. Jedoch führt das Vorgehen, jede Architekturstufe bezüglich jedes Qualitätsmerkmals und Teststufe zu testen, zu einer sehr großen Anzahl von Testfällen. Zumal die resultierenden Testfälle nicht immer unbedingt sinnvoll erscheinen. Gut wäre eine Zuordnung von Architekturebene zu Teststufen. Wie eine Testautomatisierung in eine zeitlich begrenzte Iteration integriert werden kann, wird nicht betrachtet.

	Bewertung
Organisation	
Prozessdefinition	[ ]
Rollendefinition	[ ]
Agil Iterativ	[ ]
Teststufen	[+] Das Gesamtsystem wird anhand der Architektur in Subsysteme aufgeteilt. Pro Subsystem werden die Teststufen durchgeführt. Eine Verteilung der Testfälle auf die einzelnen Teststufen findet nicht statt.
BI – Architektur	[++] Das BI – System wird für das Testen in Subsysteme eingeordnet. Die Qualitätsmerkmale werden auf die Systemteile verteilt.
Methoden	

Testfallentwurf	[ ]
Test Priorisierung	[+] Das Testen des Zentralen Datenbestands wird priorisiert. Verfahren eine projektindividuelle Priorisierung durchzuführen, werden nicht erwähnt.
Metriken	[ ]
Automatisierung	[+] Eine Einordnung welche Testfälle automatisiert werden können, findet anhand von Qualitätsmerkmalen und Testobjekten statt.
Legende: [ ] nicht Beschrieben [-] nicht geeignet [+] wird berücksichtigt [++] wird vollumfassend berücksichtigt	

Tabelle 5.4 Bewertung der BI Teststrategie von (Krawatzeck, Zimmer, & Trahasch, 2013)

5.3 Collier, 2012

5.3.1 Methode und logische Testfälle

Collier beschreibt vier Dimensionen des Testens. Die Dimensionen Geschäftsakzeptanz *(Business acceotability*) und Produktvalidität *(Product validation)* sind Geschäfts- bzw. Endnutzer zentriert und dienen der kritischen Betrachtung des Produktes. Diese Dimensionen stellen sicher, dass das richtige Produkt erstellt wird. Die weiteren Dimensionen technische Akzeptanz *(Technical acceptability)* und Systemvalidierung *(System validation),* sind Technologie bzw. Entwickler zentriert. Sie überprüfen die technisch richtige Umsetzung des Systems. Die Dimensionen bilden eine Matrix, anhand welcher sich Teststufen und Automatisierungsgrade zuordnen lassen.

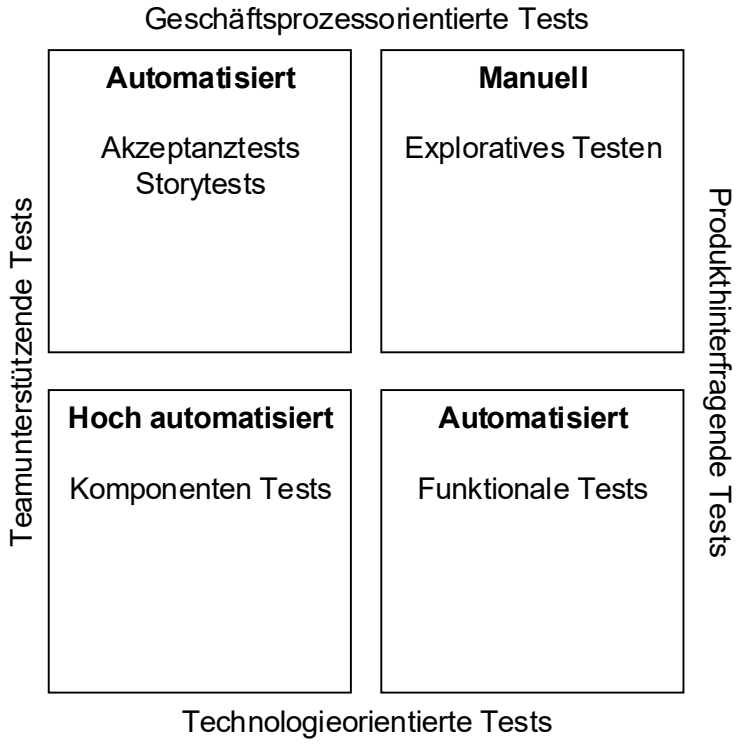

Abb. 5-1 2x2 Testmatrix

TESTSTUFEN

Die Teststufen sollen einmal je Iteration ausgeführt werden. Wobei Unit Tests kontinuierlich während der Entwicklung stattfinden. Story Tests sind eher Product Owner getrieben. Usability Tests finden typischerweise eher am Ende der Iteration statt.

Unit Tests

- Integrationstests („Story testing“)
- Funktionale Tests
- Exploratives Testen

Collier empfiehlt zum Ende jedes Sprints oder Iteration die Durchführung explorativer Testfälle. Durch diese manuelle Form des Testens soll aufgezeigt werden, wie „echte Nutzer“ mit dem System interagieren, und Testfälle gefunden werden, die vorher nicht für die Automatisierung in Betracht gezogen wurden.

TESTZIELE

Die Matrix wird durch verschiedene Teststrategien für Performance, Load und Stresstests ergänzt. Diese Tests finden auf Systemebene statt. Sie können regelmäßig automatisiert durchgeführt werden.

PERFORMANCE TESTS

Im BI Kontext liegt der Fokus des Performance Testens auf der Verarbeitung großer Datenmengen. Hierfür empfiehlt Collier das Erstellen eines möglichst großen Testdatensatzes, der mit dem Datenvolumen im Produktivsystem vergleichbar ist. Zudem sollten mögliche Engpässe in der Applikation, dem Betriebssystem und den netzwerkschichten der Architektur betrachtet werden.

Load Test

Für das „Load Testing“ wird die Reaktion des Systems, bei hohen Zugriffszahlen getestet. Hierfür werden jedoch spezielle Tools benötigt.

Stresstest

Im Rahmen des Stresstest wird das System bis zu dem Punkt überladen, an dem es versagt. Stress Test könne auch das Testen unerwarteter Ereignisse bedeuten. Ein Beispiel wäre das zufällige Trennen einer Datenbank, oder das Abbrechen von Ladevorgängen.

TESTOBJEKTE ANHAND DER ARCHITEKTUR

Collier identifiziert die Testpunkte entlang der Datenflüsse ausgehend von den operativen Systemen.

- Operative Datenbank
- Datenupdate
- Datenaufbereitung
- Datentransformation
- Datenverteilung
- Datenzugriffsschicht
- BI Applikation
- Administrative Applikation

5.3.2 Daten

Das Set von Testdaten sollte so klein wie möglich, gleichzeitig jedoch ein repräsentativer Auszug der aktuellen produktiven Daten sein. Zudem sollte das Set die möglichen Werte für verschiedene Datenelemente abbilden. Um ein solches Datenset zu erhalten, schlägt er vor für jedes Feld den größten, den kleinsten, und einen durchschnittlichen Wert zu selektieren. Um Grenzfälle zu testen, sollten einige negative Werte ausgewählt werde. Eine Erweiterung des Datensets ist nur bei neuen Anforderungen vorgesehen. Es wird also das ganze Projekt mit dem gleichen Datenset getestet. Den Einsatz von Produktivdaten sieht er kritisch. Sollten jedoch Produktivdaten für das Testen verwendet werden, müssen sensitive Informationen ersetzt werden.

Korrespondierend zu den Testdaten existiert ein Datensatz mit erwarteten Ergebnissen (*expected results set*) für die Verifizierung der Ergebnisse. Wobei genau definiert ist, was die Ergebnis Daten umfassen und was nicht. Wenn die Testdaten erweitert werden, muss analog dazu auch das

Ergebnisset erweitert werden. Weil die Testdaten so erstellt wurden, dass sie ein minimal ausreichendes Set an Daten darstellen, ist es kritisch jedes Ergebnis zu überprüfen. Eine auszugsweise Überprüfung der Daten ist nicht ausreichend. Jedes erwartete Ergebnis muss geprüft werden, und es dürfen keine unerwarteten Ergebnisse auftreten.

Damit die Ergebnisse eines Testfalls nicht das Verhalten anderer Tests beeinflusst, wird das Testsystem nach jedem Test wieder in seinen Ausgangszustand gebracht. Jeder Testfall sollte so isoliert wie möglich ausgeführt werden, sodass Ergebnisse zuverlässig und wiederholbar sind. Probleme können so einfach analysiert und behoben werden.

5.3.1 Bewertung

Collier betrachtet das BI-Testing aus der Sicht BI-Agilität. Er ordnet die Teststufen sinnvoll in einen iterativen Entwicklungszyklus ein. Wobei entwicklernahe technische Testfälle häufiger durchgeführt werden, als höhere Integrationsstufen. Zeitintensive Tests werden außerhalb der eigentlichen Entwicklungsiteration durchgeführt. Dies entspricht den Anforderungen an eine Testverteilung im agilen Sinne. Anhand der Testmatrix lässt sich eine minimale Rollenverteilung ableiten. Technisch orientierte Tests werden durch den Entwickler durchgeführt, wohingegen fachliche Tests Product Owner getrieben sind. Die Matrix dient auch als Orientierung für die Testautomatisierung. Colliers Ansatz zum Erstellen von Testdaten entspricht einer Äquivalenzklassenbildung in Kombination mit einer Grenzwertanalyse. Der dadurch entstehende minimale Umfang der Testdaten wirkt sich positiv auf die Laufzeit der Testfälle aus und ermöglicht eine schnelle Rückmeldung der Testergebnisse. Metriken, die eine Testabdeckung abbilden, werden nicht explizit genannt.

	Bewertung
Organisation	
Prozessdefinition	[+]
Rollendefinition	[+] Anhand der 2x2 Matrix lässt sich eine einfache Rollendefinition und eine entsprechende Zuordnung zu Testverfahren ableiten.
Agil Iterativ	[++] Der gesamte Prozess ist auf agile Entwicklungsmethoden ausgerichtet.
Teststufen	[++] Die Teststufen werden anhand der 2x2 Matrix auf die Iteration verteilt.

BI – Architektur	[++] Die Besonderheiten der BI-Architektur werden berücksichtigt. Insbesondere das benötigte Testdatenset wird beschrieben.
Methoden	
Testfallentwurf	[ ]
Test Priorisierung	[ ]
Metriken	[ ]
Automatisierung	[+] Eine Zuordnung von Testautomatisiert zu Integrationsstufen findet anhand der 2x2 Matrix statt. Testautomatisierung ist in Form von Regressionstest wesentlicher Bestandteil der Teststrategie.
Legende: [ ] nicht Beschrieben [-] nicht geeignet [+] wird berücksichtigt [++] wird vollumfassend berücksichtigt	

Tabelle 5.5 Bewertung der BI Teststrategie nach (Collier, 2012)

5.4 Neveen ElGamal, 2013

METHODE UND LOGISCHE TESTFÄLLE

ElGamal's Ziel ist es eine DW-Test Framework zu entwickeln. Hierzu nimmt er eine Klassifizierung von logischen Testfällen anhand daran „wo", „was" und „wann" getestet wird, vor. Mit „wo" meint die Komponenten des DHWs anhand der Architektur. Explizit nennt er folgende Testpunkte:

- Datenquellen und operationale Data Stores
- Ladeprozess aus den Quellen
- ETL Prozess in die Data Marts
- UI Anbindung an die Data Marts

Für das „Was" nennte er drei zu testenden Aspekte. Zum einen die Datenbank Schema, mit Fokus auf dem Design. Zum anderen die Daten, mit Fokus auf Datenqualität, Datentransformationen, Datenauswahl und Datenpräsentation. Der dritte Aspekt ist der Test der Inbetriebnahme der Software. Daran schließt auch das „Wann" an. Hier unterscheidet ElGamal zwischen der Zeit vor, und nach der Auslieferung. Durch die Anordnung der drei Dimensionen zu einer Matrix können logische Testfälle zugeordnet werden.

		Schema	Daten	Be-trieb
Backend	Data Store -> Operational Data Store			
	Operational Data Store -> Data-Warehouse			
Frontend	Data-Warehouse -> Data Mart			
	Data Mart -> User Interface			

Tabelle 5.6: Zuordnung Testaspekte und Testpunkte

5.4.1 Bewertung

	Bewertung
Organisation	
Prozessdefinition	[ ]
Rollendefinition	[ ]
Agil Iterativ	[ ]
Teststufen	[ ]
BI – Architektur	[+] Das Testobjekt wird anhand der Architektur in kleinere handhabbare Teile zerlegt.
Methoden	
Testfallentwurf	[+] Der Testfallentwurf berücksichtigt die Architekturebene und den Zeitpunkt im Testverlauf. Anhand dieser Dimensionen können Testfälle abgeleitet werden.
Test Priorisierung	[ ]
Metriken	[ ]
Automatisierung	[ ]
Legende: [ ] nicht Beschrieben [-] nicht geeignet [+] wird berücksichtigt [++] wird vollumfassend berücksichtigt	

Tabelle 5.7 Bewertung der BI Teststrategie nach (Neveen ElGamal, 2013)

5.5 Rizzi & Golfarelli, 2009

Für die Klassifizierung logischer Testfälle betrachten Rizzi & Golfarelli was und wie getestet wird, wobei die Datenqualität im Mittelpunkt steht. Datenqualität bedeutet für Rizzi & Golfarelli im Wesentlichen die Richtigkeit der Daten, die von den ETL Prozessen geladen und von Frontend Tools aufgerufen werden. Jedoch wird der Qualität des Designs die gleiche Priorität beigemessen. Dies bedeutet im Detail, dass Nutzeranforderungen durch das konzeptuelle Schema des Data Marts repräsentiert werden. Den Testpunkten stellen sie Testarten gegenüber (wie).

	Analyse & Design		**Implementation**		
	Konzeptuelles Schema	**Logisches Schema**	**ETL Prozesse**	**Datenbanken**	**Front End**
Funktionale Tests	x	x	x		x
Usability Tests	x	x	x	x	x
Performance Tests			x	x	
Stresstests			x	x	x
Regression Tests	x	x	x	x	x

Tabelle 5.8 Zuordnung Testpunkte zu Teststufen (Rizzi & Golfarelli, 2009)

METRIKEN

Als wichtigste Voraussetzung für einen effektiven Test nennen Rizzi & Golfarelli die frühe Definition von notwendigen Bedingungen Das bedeutet, dass Metriken gemeinsam mit Akzeptanzkriterien eingeführt werden müssen. Quantifizierbare Metriken sind auch für das automatisierte Testen notwendig. Für einige Testarten, wie zum Beispiel der maximalen Antwortzeit, können Metriken aus der Softwareentwicklung verwendet werden, für andere Testarten werden DWH spezifische Metriken benötigt. Für die Definition solcher Metriken schlagen Rizzi & Golfarelli fünf Schritten vor:

- Ziel identifizieren, indem die darunterliegende Hypothese spezifiziert, und die Qualitätskriterien gemessen werden.
- Die Metriken Formal definieren.
- Metriken theoretisch validieren, z.B. axiomatisch.
- Metriken empirisch validieren, indem die Metriken auf Daten anderer Projekte angewendet werden.

- Anwenden

Das Konzeptuelle Schema Testen

Je früher Fehler im Software Lebenszyklus entdeckt werden, desto günstiger ist es, diese Fehler zu korrigieren. Deshalb sollte eine Testmethode für Data Warehouse Systeme, das konzeptuelle Schema mit einbeziehen. Für das konzeptuelle Schema bieten sich zwei Testtypen an: zum einen der Faktentest, eine Verifikation anhand der funktionalen Anforderungen: Sind die erforderlichen Kennzahlen, Dimensionen und Aggregationen vorhanden? Zum anderen der Konformitätstest: Hier wird überprüft, inwiefern die Hierarchien dem abzubilden Geschäftsmodell entsprechen(Rizzi & Golfarelli, 2009, S. 4).

Testen des logischen Schemas

Vor der Implementation kann die Anzahl der Fehler, die durch schlechtes logisches Design verursacht werden, durch Tests reduziert werden. Dies kann zum Beispiel durch ein Sample von Abfragen auf dem logischen Schema passieren. Dieser „*Star Test*“ prüft, ob Abfragen korrekt in SQL formuliert und auf dem logischen Schema ausgeführt werden können.

ETL Testen

Das ETL Testen ist die komplexeste und kritischste Testphase, mit direktem Einfluss auf die Daten. Zum Validieren der Tests können jedoch Standard Techniken aus der Software Entwicklung verwendet werden. Unit Tests zum Prüfen des ETLs können entweder vertikal oder horizontal an der Architektur ausgerichtet sein. Bei vertikalen Testfälle existiert ein Testfall für jede Dimension und eine Dimension für jede Gruppe von korrelierenden Fakten. Horizontale Testfälle schneiden die Tests entlang der Architekturebene. Integrationstest testen wiederum die Datenflüsse der ETL Prozesse. Zum Testen der Qualitätskriterien (1) Datenintegrität, (2) Vollständigkeit und (3) Aktualität empfehlen Rizzi & Golfarelli die Nutzung von drei Datenbanken mit unterschiedlichen Datenausprägungen:

- Korrekte und vollständige Daten
- Fehlerhafte Daten
- Real Daten

Performanz und Stress Test sind Komplementär um die Effizienz der ETL Prozeduren zu testen. Performanztest evaluieren das Verhalten des ETL Prozesses bei Standardroutinen, im speziellen prüfen sie, ob die Verarbeitungszeit mit dem erwarteten Zeitfenster für den Staging Prozess übereinstimmt. Stresstest simulieren außergewöhnliche Aufkommen an Daten. Als weitere nicht funktionale Tests nennen Rizzi & Golfarelli Recovery und Sicherheitstests (Rizzi & Golfarelli, 2009, S. 21).

5.5.1 Bewertung

Die Zuordnung zwischen Teststufen und Architekturebenen stellt eine gute Einordung für das Erstellen möglicher Testfälle dar. Insbesondere, dass Einbeziehen der Analyse und Designphase in die Betrachtung, kann dazu führen, dass im weiteren Projektverlauf weniger Fehler auftreten. Das Vorgehen bereits in der Designphase Metriken zu definieren, ermöglicht den Umfang der Tests festzulegen und auch eine Definition of done zu beschrieben. Eine Vergleichbarkeit der System Qualität ist so gegeben.

	Bewertung
Organisation	
Prozessdefinition	[+]
Rollendefinition	[ ]
Agil Iterativ	[ ]
Teststufen	[++] Die Teststufen werden den Architekturebenen zugeordnet.
BI – Architektur	[++] Testfälle und Architekturebenen werden einander Zugeordnet und architekturspezifische Testfälle definiert.
Methoden	
Testfallentwurf	[++] Der Testfallentwurf findet anhand der Qualitätskriterien statt.
Test Priorisierung	[+]
Metriken	[++] Es werden Metriken definiert, und eine Vorgehensweise zum ermitteln eigener Metriken vorgestellt.
Automatisierung	[++]
Legende: [ ] nicht Beschrieben [-] nicht geeignet [+] wird berücksichtigt [++] wird vollumfassend berücksichtigt	

Tabelle 5.9 Bewertung der BI Teststrategie nach (Rizzi & Golfarelli, 2009)

5.6 Hughes, 2016

Hughes beschreibt ein Framework für agiles Qualitätsmanagement im BI-Umfeld. Dabei verfolgt er drei grundlegende Prinzipien:

- Nach Balance streben
- Inkrementelles Vorgehen

- Test getriebene Entwicklung

Als das Streben nach Balance definiert Hughes die risikogetriebene Testauswahl. Jedes Modul wird gerade so („just enugh“) getestet, sodass sichergestellt ist, dass es den Anforderungen entspricht, aber andererseits nicht den Zeitplan des gesamten Projekts gefährdet. Das zweite Prinzip, auf dem sein Framework beruht, sind die agilen Prinzipien und Methoden. Daraus folgt, dass die Testausführung mit der ersten Iteration beginnt und alle Stakeholder des Projekts von Beginn an in den Testprozess eingebunden sind. Als Methode für das Erstellen von Testfällen beschreibt Hughes die testgetriebene Entwicklung. Dabei erweitert er das Prinzip über das Testen von Komponenten hinaus auf alle Teststufen. Dieses Vorgehen führt zu einem frühzeitigen Auseinandersetzen mit den Anforderungen, nicht nur auf Komponentenebene (Hughes, 2016). Organisatorisch teilt Hughes die Qualitätssicherung in drei Ebenen. Die Ebenen sind jeweils einer technischen Integrationsebene des Testobjektes zugeordnet.

Testobjekt	**Organisatorische Ebene**	**Integrationsebene**
Geschäftsprozesse	Qualitätsmanagement	End-to-end Test Systemtest
Applikation	Qualitätssicherung	Integrationstest
Komponenten	Qualitätskontrolle	Komponententest

Tabelle 5.10 Zuordnung: Testobjekt - organisatorische Ebene

Für die Zuordnung von Testfällen zu einer Teststufe nutzt Hughes die gleiche Methode wie Collier. Anhand der 2x2 Matrix werden die Testfälle Integrationsebenen zugeordnet und die Automatisierbarkeit definiert. Gleichzeitig verfolgt er einen Bottom-Up Ansatz indem er Datawarehouse spezifische Testtechniken auf Komponentenebene beschreibt. Aus diesen Techniken können Entwickler dann projektindividuelle Testfälle erstellen.

BI-Spezifische Testtechniken

- Testen einer Data Warehouse Tabelle
- Referenzielle Integrität
- Corner Test
- Erwartete Ergebnisse testen

Diese Ansätze ergänz Hughes um das explorative Testen als Technik, die nicht nur von Entwicklern ausgeführt werden kann, sondern auch auf Systemeben und während Produktdemonstrationen am Ende der Iteration eingesetzt werden kann. Rollen und Verantwortlichkeiten ordnet Hughes anhand des V-Modells zu.

5.6.1 Bewertung

Hughes stellt den umfassendsten Ansatz für das agile BI – Testen vor. Hierbei geht er auf die Besonderheiten des Iterativen Prozesses ein. Die Verteilung der Testfälle über die Teststufen erfolgt nach dem Beispiel von Cohn (Cohn, 2009). Dabei achtet er darauf, dass der Prozess leichtgewichtig bleibt. Bei der Auswahl der Testpunkte unterscheidet sich sein Ansatz nicht von den anderen Autoren. Eine Besonderheit ist die umfassende Testfallbibliothek.

	Bewertung
Organisation	
Prozessdefinition	[++] Eine Einordnung der Prozessschritte erfolgt anhand der 2x2 Matrix. Die Anordnung der Teststufen innerhalb der Iteration wird beschrieben.
Rollendefinition	[++] Eine Rollenzuordnung erfolgt anhand des V-Modells. Zusätzlich wird eine zentrale Tester Rolle beschrieben.
Agil Iterativ	[++] Das Konzept baut auf agilen Methoden auf. Der Prozess wird auf Iterationsebene beschrieben.
Teststufen	[++] Eine Einordnung der Prozessschritte erfolgt anhand der 2x2 Matrix. Die Anordnung der Teststufen innerhalb der Iteration wird beschrieben.
BI – Architektur	[++] Die BI – Architektur wird Anhand der Architekturebenen und Datenflüsse aufgeteilt. Es werden spezifische Testmethoden je Architekturebene genannt.
Methoden	
Testfallentwurf	[++] Hughes liefert eine umfassende Bibliothek von möglichen Testfällen auf Komponentenebene.
Test Priorisierung	[ ]
Metriken	[ ]
Automatisierung	[++] Es wird ein umfassendes Konzept für die Testautomatisierung vorgestellt.
Legende: [ ] nicht Beschrieben [-] nicht geeignet [+] wird berücksichtigt [++] wird vollumfassend berücksichtigt	

Tabelle 5.11 Bewertung der BI Teststrategie nach (Hughes, 2016)

5.7 Zusammenfassende Bewertung

Bezogen auf die Ausgangsfrage dieser Arbeit, wie sichergestellt werden kann, dass bei einer iterativen Erweiterung des BI-Systems bestehende Komponenten ohne manuellen Retest weiterhin funktionieren, ergeben sich zwei Kernanforderungen für eine Teststrategie. Zum einen muss der methodische Aspekt, automatisierte Regressionstests in den Iterativen Entwicklungszyklus zu integrieren erfüllt sein. Zum anderen muss die Funktionsfähigkeit des bestehenden Systems quantifiziert werden. Der Fokus der vorgestellten Testframeworks liegt mehrheitlich auf den methodisch technischen Aspekten. Die organisatorischen Rahmenbedingungen für das Integrieren des Testprozesses können zum Beispiel aus der Methodik von Hughes und Collier abgeleitet werden. Die sinnvolle Zuordnung von Prozessschritten und Testwerkzeugen ergibt sich aus den Ausführungen von (Stauffer, Honegger, & Gisin, 2013). Die Besonderheiten der BI-Architektur werden bei allen Autoren berücksichtigt. Kernaspekt ist hier vor allem das Zerlegen der Architektur in einzelne Komponenten und das Testen in handhabbaren Teilen. Auf Integrationsebene steht das Testen der Datenflüsse entlang der Architekturebenen im Fokus. Tabelle 5.12 zeigt die zusammenfassend Schwerpunkte der einzelnen Teststrategien.

	(Stauffer, Honegger, & Gisin)	**(Krawatzeck, Zimmer, & Trahasch)**	**(Collier)**	**(Neveen ElGamal)**	**(Rizzi & Golfarelli)**	**(Hughes)**
Organisation						
Prozessdefinition	[++]	[ ]	[+]	[ ]	[+]	[++]
Rollendefinition	[++]	[ ]	[+]	[ ]	[++]	[++]
Agil Iterativ	[-]	[ ]	[++]	[ ]	[++]	[++]
Teststufen	[+]	[+]	[++]	[ ]	[++]	[++]
BI – Architektur	[++]	[++]	[++]	[+]	[++]	[++]
Methoden						
Testfallentwurf	[+]	[ ]	[ ]	[+]	[++]	[++]
Test Priorisierung	[+]	[+]	[ ]	[ ]	[+]	[ ]
Metriken	[ ]	[ ]	[ ]	[ ]	[++]	[ ]
Automatisierung	[-].	[+]	[+]	[ ]	[++]	[++]

Legende: [ ] nicht Beschrieben [-] nicht geeignet [+] wird berücksichtigt [++] wird vollumfassend berücksichtigt

Tabelle 5.12 Einordnung der BI Teststrategien

6 BI TESTSTRATEGIE

In diesem Kapitel wird anhand der gesammelten Erkenntnisse eine risikoorientierte Teststrategie für BI-Systeme erarbeitet. Methodisch orientiert sich die Teststrategie an den durch Scrum vorgegeben Meetings und Rollen. Die Sprintlänge beträgt 30 Kalendertage. Scrum dient dabei nur als Beispiel einer agilen Vorgehensweise. Die Umsetzung der Teststrategie muss unternehmens- und projektindividuell erfolgen. Als Struktur für das Zieldokument, aber auch als roter Faden für die Erarbeitung der Strategie dient die Teststrategie nach IEEE 829.

6.1 Organisation

6.1.1 Prozess

Ausgangspunkt für den Aufbau einer Iteration ist der definierte Ablauf eines Scrum Sprints. Zu Beginn der Iteration werden während der Sprintplanung Aufgaben aus dem priorisierten Product Back Log ausgewählt und in den Sprint Back Log aufgenommen. Dieses Arbeitspaket (Inkrement) wird während der laufenden Iteration nicht durch Zusatzanforderungen modifiziert. Während des Sprints arbeitet das Team daran, die Aufgaben aus dem Sprintbacklog umzusetzen. Ziel jedes Sprints ist es, am Ende des Sprints ein potenziell auslieferbares Produkt Inkrement zu liefern. Am Ende des Sprints präsentiert das Team im Sprint Review die implementierte Funktionalität. Diese Meetings / Prozessphasen dienen als Ausgangspunkt für die Einordnung der Phasen des Testprozesses. Sowohl Collier als auch Hughes [(Hughes, 2016) (Collier, 2012)] geben Anhaltspunkte für die Einordnung der Testphasen. Abb. Zeigt den schematischen Ablauf einer Entwicklungsiteration.

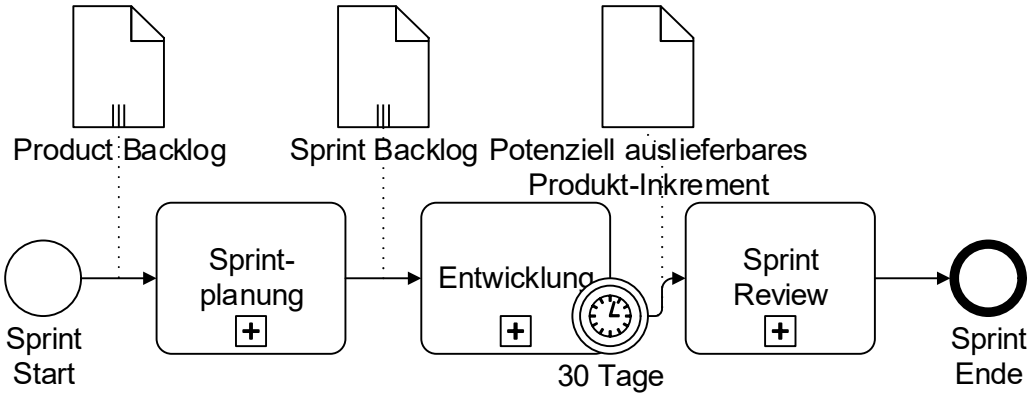

Abb. 6-1 Aufbau einer Scrum Iteration

SPRINT NULL

Die Aufgaben zur Qualitätssicherung sollten also in diesen Ablauf integriert werden. Der allgemeine Testprozess enthält jedoch auch Aufgaben, die dem Prozess vor- bzw. nachgelagert erfolgen müssen. Die Aufgaben der Testplanung beziehen sich auf das gesamte Projekt und müssen somit vor dem ersten Sprint abgeschlossen sein. Deshalb wird vor dem ersten Entwicklungssprint ein Sprint null durchgeführt, in dem alle Projektvorbereitenden Aufgaben erledigt werden.

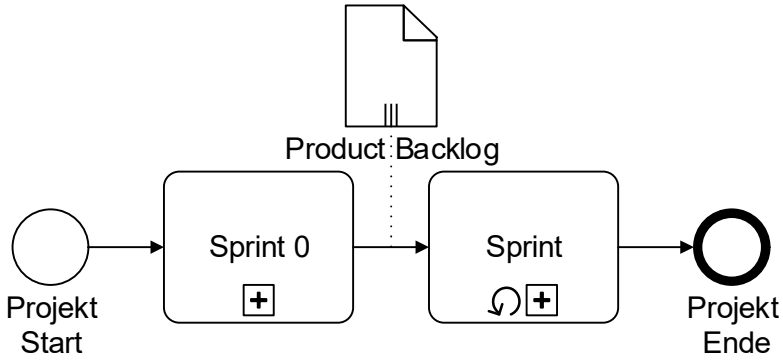

Abb. 6-2 Einordnung Sprint Null

Ergebnis dieser Phase ist ein Ressourcen Plan, das fertige Testkonzept als Basis für die Teststeuerung und eine erste Risikoabschätzung anhand der Zielarchitektur. Die Planung der Testautomatisierung ist Bestandteil dieser Phase. Die Ergebnisse der Testplanung fließen in die Analyse und Designphase jeder Iteration mit ein.

Teststufe	**Input**	**Output**
Testplanung	▪ Teststrategie ▪ Product Backlog ▪ Architekturdokumente	▪ Testkonzept ▪ Risikoinventar ▪ QM Ressourcenplan

Tabelle 6.1 Beschreibung des Prozessschritts Testplanung

ENTWICKLUNGSSPRINT

Aus der zeitlichen Begrenzung der Iteration auf 30 Tage entsteht die Herausforderung, die Testaufgaben trotzdem mit der nötigen Intensität durchzuführen. Dazu werden drei Maßnahmen genutzt. Zum einen die risikoorientierte Auswahl von Testfällen, zum anderen die Verteilung der Testfälle auf die Integrationsstufen und die Ergänzung automatisierter Regressionstest. Die Risikoidentifizierung & Risikobewertung erfolgt während der Sprintplanung anhand des Sprintbacklogs und dem vorläufigen Risikoinventar als Ergebnis der Testplanung. Anhand der Risikobewertung werden während der Testspezifikation, Testfallentwurfsmethoden für die einzelnen Teststu-

fen ausgewählt. Aus den Ausführungen von Collier und Hughes können die Teststufen, die Häufigkeit und der Zeitpunkt der Testdurchführung innerhalb der Iteration abgleitet werden [(Hughes, 2016), (Collier, 2012)].

- Komponententests [Kontinuierliche Testdurchführung]
- Integrationstest [Kontinuierliche Testdurchführung]
- Funktionales Testen [Am Ende der Iteration]
- Exploratives Testen [Am Ende der Iteration]

Als Anhaltspunkt für die Verteilung der Testfälle über die Teststufe dient der Grundsatz von Cone, technische Überprüfungen kontinuierlich durchzuführen. Relativ einfach zu automatisierende Komponententests mit kurzer Laufzeit können oft und aufwandsarm durchgeführt werden. Deshalb werden die Komponenten kontinuierlich parallel zur Entwicklung getestet. Ergänzend werden täglich Integrationstests durchgeführt. Integrationstests sind ähnlich einfach zu automatisieren wie Komponenten Tests, haben jedoch eine längere Laufzeit. Fachlich orientierte Tests werden, aufgrund der benötigten hohen Integrationsstufe, am Ende der Iteration durchgeführt. Ziel ist jedoch die fachlichen Tests bereits auf der kleinsten möglichen Integrationsstufe durchzuführen. Kann ein fachlicher Test also bereits auf Data Mart Ebene erfolgen, ist dies zu bevorzugen. Oberflächentests sind im Verhältnis zu datengetriebenen Testfällen komplexer zu automatisieren und haben eine längere Laufzeit. Die fachlichen Regressionstests erfolgen aufgrund ihrer Laufzeit einmal wöchentlich am Wochenende. Explorative Tests ergänzen die Teststufen und tragen dazu bei mögliche Fehler und systematische Schwächen der methodisch aufgebauten Teststrategie zu vermeiden. Daraus ergibt sich das in .Abb. 6-3 gezeigte Ziel Bild des Testprozesses.

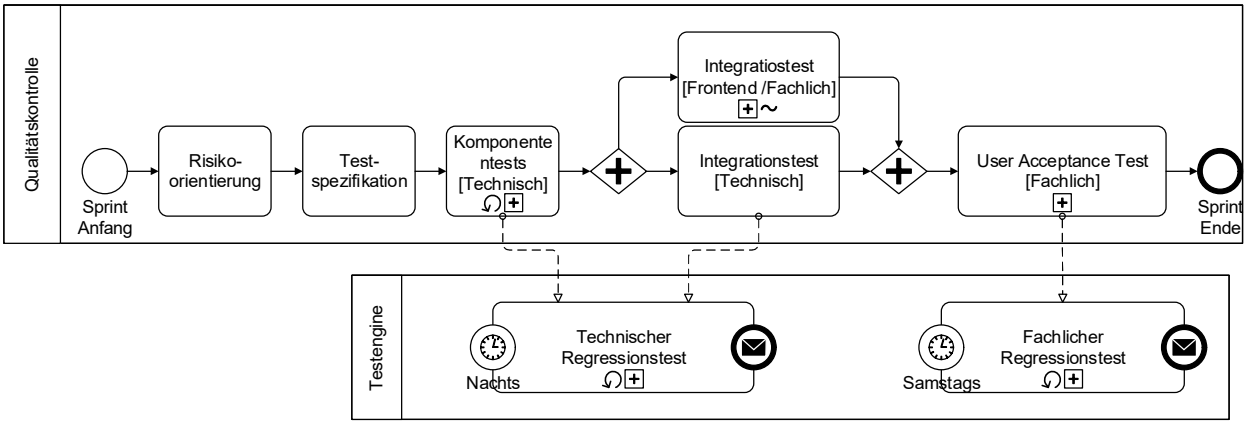

Abb. 6-3 Testschritte innerhalb der Iteration

Teststufe	Input	Output
Risikoorientierung	▪ Sprintbacklog ▪ Risikoinventar ▪ Komplexitätsmetriken ▪ Architekturdokumentation	▪ Risikoinventar (Detailliert)
Testspezifikation	▪ Risikoinventar (Detailliert) ▪ Anforderungen aus dem Sprintbacklog ▪ Architekturdokumentation ▪ Schnittstellen des Testobjektes	▪ Testfallspezifikation
Komponententests	▪ Testfallspezifikation	▪ Komponententestfälle ▪ Testergebnis ▪ Komplexitätsmetriken
Integrationstest [Fachlich]	▪ Testergebnis Komponententests ▪ Testfallspezifikation	▪ Integrationstestfälle ▪ Testergebnis
Integrationstest [Technisch]	▪ Testergebnis Komponententests ▪ Testfallspezifikation	▪ Testergebnis
User Acceptance Test	▪ Testergebnisse der Integrationstests	▪ Testergebnis
Technischer Regressionstest	▪ Komponententestfälle ▪ Integrationstestfälle	▪ Testergebnis
Fachlicher Regressionstest	▪ Testergebnis des fachlichen Integrationstest	▪ Testergebnis

Tabelle 6.2 Zuordnung der Teststufen

Systemtests können aufgrund ihrer Laufzeit nicht innerhalb der Entwicklungsiteration durchgeführt werden. Hughes Ansatz diese nachgelagert durchzuführen führt dazu, das System und Abnahme Tests auf der einen Seite nicht die Entwicklungsarbeit unterbrechen, aber auf der anderen Seite genügend Zeit für intensives Testen ist.

6.1.2 Rollen

Die durch Scrum definierten Rollen bilden die Minimalanforderung. Das Scrum Team ist ein sich selbst organisiertes Team. Dementsprechend gibt es auch keinen „Teilprojektleitertest“ (Linz, 2017, S. 44). Die organisatorischen Aufgaben fallen in den Aufgabenbereich des Scrummasters: Testaufgaben werden explizit über eigene Tasks geplant oder implizit als Teil der Definition of Done anderer Aufgaben. Dies erfolgt durch das Team. Für die inhaltliche Ausgestaltung der Tests und das Anpassen der Teststrategie auf die projektspezifischen Bedürfnisse bedarf es jedoch eine Person im Team, die Erfahrung als Softwaretester mitbringt. „Diese Person steuert dann ihre Expertise bei, um die Softwaretests inhaltlich fachgerecht, risikoorientiert und wirtschaftlich aufzusetzen und über alle Sprints hinweg zu realisieren.“ (Linz, 2017, S. 44) Diese Person wird im Folgenden Testmanager genannt. Fachliche Tests sind Productowner getrieben. Daraus ergibt sich ein vorläufiges Zielbild für die Rollendefinition.

Rolle	**Prozessschritte**
Scrummaster	Testplanung, Testorganisation
Testmanager	Fachliche Unterstützung bei der risikoorientierten Testfallauswahl und beim Testentwurf
Entwicklerteam	Komponententest, Integrationstest
Productowner	Fachliche Tests

Tabelle 6.3 Rollendefinition

Die heterogene Architektur eines BI-Systems führt häufig zu einer spezialisierten Rollenverteilung anhand der BI-Architektur. Grundsätzlich gilt, wie auch von Hughes und Collier propagiert, die Zuordnung des V-Modells. Die Rolle im Test Prozess leitet sich aus der Rolle im Entwicklungsprozess ab. Je nach Projektgröße kann für die Testorganisation und Steuerung eine eigenständige Rolle geschaffen, oder die Rolle des Testmanagers um organisatorische Aufgaben erweitert werden. Diese Rolle kann auch projektübergreifend sein. Zum Testen der Qualitätsmerkmale Performance, Wartbarkeit, Zuverlässigkeit, Effizienz, Wartbarkeit und Übertragbarkeit existieren in der Organisation häufig projektübergreifende Rollen, auf die für diese Tests zurückgegriffen wird.

6.2 Werkzeuge, Techniken, Methoden, Metriken

Um dem Ziel, mit möglichst wenig Aufwand möglichst viele Anforderungen zu überprüfen, bzw. Fehlerwirkungen nachzuweisen, nahezukommen, muss ein systematisches Vorgehen beim Entwurf von Testfällen gewählt werden. Als weiterer Aspekt des effizienten Testens dient die risikoorientierte Auswahl von Testobjekten. Risikoorientiertes Testen nutzt das Wissen über Risiken dazu, den Inhalt des Testkonzeptes so festzulegen, dass unter gegebenen Rahmenbedingungen die Risiken möglichst minimiert werden.

6.2.1 Testfallentwurfsmethoden anhand der Qualitätskriterien

Die zu prüfenden Qualitätskriterien ergeben sich aus der Norm ISO9126. Sie unterscheiden sich bei den Autoren nur marginal. Tabelle 6.4 Zuordnung von Qualitätskriterien zu Architekturebenen zeigt zusammenfassend, welche Architekturebenen bezüglich welchen Qualitätskriteriums getestet werden.

	Datenflüsse	**Datenhaltung**	**Datenbereitstellung**	**Front End**
Funktionalität	X	X	X	X
Benutzbarkeit				X
Wartbarkeit	X	X	X	X
Sicherheit				X
Robustheit	X		X	
Effizienz	X	X	X	X

Tabelle 6.4 Zuordnung von Qualitätskriterien zu Architekturebenen

FUNKTIONALITÄT

Das Qualitätsmerkmal Funktionalität prüft, ob die Software den spezifizierten Anforderungen entspricht. Die Funktionalität kann in allen Integrationsstufen getestet werden.

Die Funktionalität kann auf Systemebene anhand von anwendungsfallbasierten Tests unter Ergänzung von Äquivalenzklassenanalyse und Grenzwertanalyse getestet werden. Die Tests erfolgen auf UI Ebene oder, wenn möglich, auf den Datamarts.In den Integrationsstufen kann die Funktionalität mit entscheidungsbasierten Testfällen entlang der Datenflüsse geprüft werden. Der Überdeckungsgrad stellt somit die Testintensität der Schnittstellen dar.

Auf Komponentenebene können der Architekturebene entsprechende Testverfahren gewählt werden. Die Prüfung des Datenmodells kann durch das Bilden von Äquivalenzklassen in Kombination mit einer Grenzwertanalyse erfolgen. So kann die syntaktisch korrekte Implementierung einzelner Tabellen sichergestellt werden.

Zustandsbezogene Testfälle eignen sich für die ETL Prozesse. Neben dem Eingabewert hat auch der bisherige Ablauf des Systems Einfluss auf das Systemverhalten. Als Beispiel kann hier die Datenintegration dienen.

- Initialer Ladevorgang wird vollständig verarbeitet
- Korrekte Verarbeitung eines zweiten Ladevorgangs
- Erkennen und verhindern eines doppelten Ladevorgangs
- Korrektes Abarbeiten von mehreren Dateien
- Rollback bei fehlerhaften Dateien
- Zurückweisen von fehlerhaften Daten

Die methodische Testfallerstellung auf Komponentenebene wird durch einen Testfallkatalog ergänzt, sodass Entwickler je nach Projektsituation auf bestehenden Testfällen aufbauen können.

BENUTZBARKEIT

„Unter Benutzbarkeit werden die Verständlichkeit und die Benutzerfreundlichkeit des Systems geprüft.“ (Stauffer, Honegger, & Gisin, 2013) Die Prüfung der Benutzbarkeit erfolgt auf User Interface Ebene. Geeignetes Testentwurfsverfahren hierfür ist ein Usability Test, bei dem die Hauptanwendungsfälle des Systems durchgespielt werden.

ZUVERLÄSSIGKEIT

Die Zuverlässigkeit kann mit den Merkmalen Verfügbarkeit und Robustheit umschrieben werden. Getestet wird die Software in verschiedenen Alltagssituationen. Mit diesen Tests soll ein stabiles System garantiert werden. Zum Überprüfen der Zuverlässigkeit die Metrik „Anzahl von erst nach Änderungen fehlgeschlagenen Testfällen“ (FWP) herangezogen werden.

EFFIZIENZ

Die Effizienz testet Punkte wie Performance und Ressourcenverbrauch. Sie spielt für die funktionale Richtigkeit des Systems keine Rolle. Getestet werden können die Antwortzeiten und der Ressourcenverbrauch des Systems.

WARTBARKEIT

Die Wartbarkeit wird mit Hilfe eines Walkthroughs getestet. Ziel ist es, die Lösung auf die Verständlichkeit der Struktur hin zu überprüfen.

ÜBERTRAGBARKEIT

Die Übertragbarkeit muss durch Reviews oder andere statische Qualitätsmaßnahmen geprüft werden. Zudem kann der FWP Wert (die *Anzahl von erst nach Änderungen fehlgeschlagenen Testfällen*) als geeignete Metrik zum Messen der Anpassbarkeit als Teilmerkmal der Übertragbarkeit herangezogen werden.

6.2.2 Risikoorientierung

Im Rahmen agiler Vorgehensweisen werden Qualitätsrisiken meist sehr informell behandelt. Eine Risikobetrachtung findet ausschließlich beim explorativen Testen während der Testdurchführung statt. Ergebnisse fließen also nicht unmittelbar in die laufenden Entwicklungsaktivitäten ein. Deshalb ist es sinnvoll, bereits bei der Sprintplanung mögliche Produktrisiken zu identifizieren und zu einem Risikoinventar zusammenzustellen. In der Retrospektiven können diese dann abgeglichen und für die nächsten Sprints aktualisiert werden. Ausgangspunkte können zum Beispiel Komplexitätsmetriken sein. Die durch Scrum vorgegebenen Methoden zur Risikobeherrschung, sind durch Ansätze zum risikoorientierten Testen zu ergänzen. Hierzu können auch pragmatische leichtgewichtige Ansätze wie PRAM, SST und PRISMA zum Einsatz kommen. Die sieben Schritte zum risikoorientierten Test nach PRAM sind im folgenden Prozess skizziert. Die Schritte der Risikobewertung und Risikoidentifizierung sind am Anfang jeder Iteration, während der Sprintplanung, durchzuführen.

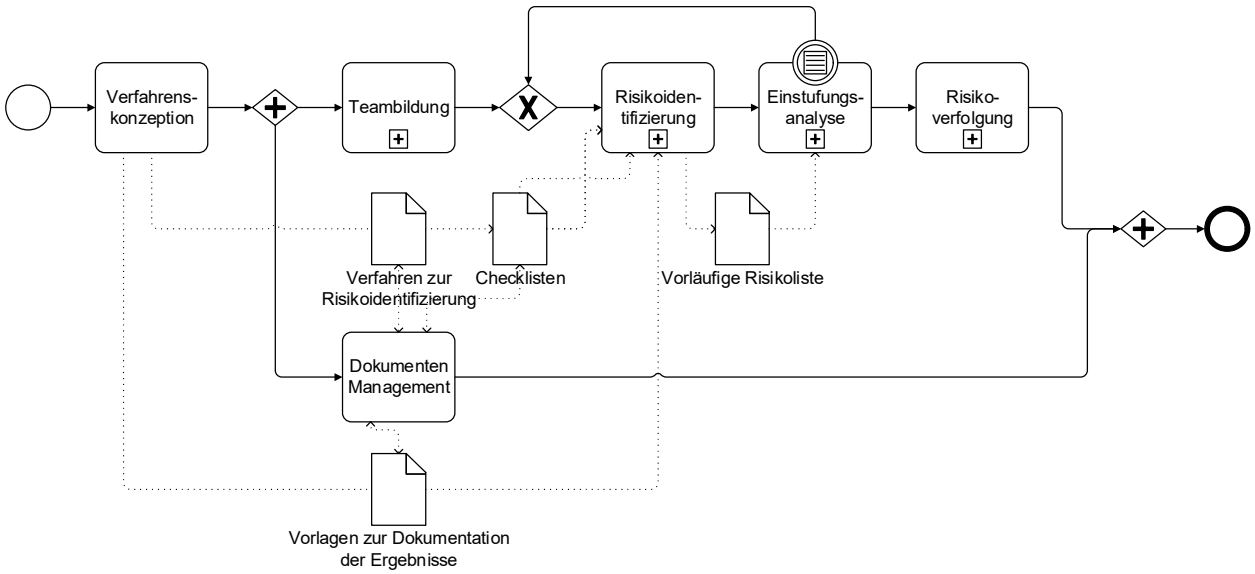

Abb. 6-4 Die sieben Schritte zum risikoorientierten Teste nach PRAM (Spillner, Roßner, Winter, & Linz, 2014, S. 142)

6.2.3 Metriken

Bezogen auf die Ausgangsfrage dieser Arbeit, die Produktqualität bei iterativer Entwicklung sicherzustellen, liegt der Fokus auf testfall- und testobjektbasierten Metriken. Ergänzend fließt das Ergebnis der Risikobewertung in die Auswahl der Metriken ein.

TESTFALLBASIERTE METRIKEN

Testfallbasierten Metriken werden für die bestehenden, geänderten und neuen Testfälle separat erhoben. Sodas zum einen die Qualität des bestehenden Produktes, und zum anderen neue und geänderte Produktteile geprüft werden können. Das zu priorisierende Ziel ist die Qualität des Testobjektes. Eine hohe *Anzahl von Testfällen, die noch nie eine Fehlerwirkung aufdeckten* (PNF) bei gleichzeitig hoher Testüberdeckung weist auf das Erreichen dieses Ziels hin. Offene Abweichungen werden durch die Metrik *Anzahl der Testfälle, die immer mit Fehlerwirkung durchgeführt werden* (FNP) angezeigt. Diesen Wert gilt es transparent zu kommunizieren, um den Status der Fehlerbehebung darzustellen. Das Qualitätsmerkmal Wartbarkeit und Änderbarkeit wird durch die *Anzahl von erst nach Änderungen fehlgeschlagenen Testfällen* (FWP) abgebildet. Dieses Merkmal ist mit Hinblick auf einen iterativen Prozess mit Priorität zu betrachten. Eine schlechte Wartbarkeit führt zu Fehlern in Folgeiterationen.

FNP (failed, never passed)	Anzahl der Testfälle, die immer mit Fehlerwirkung durchgeführt werden.
PNF (passed never failed)	Anzahl der Testfälle, die noch nie eine Fehlerwirkung aufdeckten.
FWP (failed, was passed)	Anzahl aktuell fehlgeschlagener Testfälle, die vor der Änderung ohne Fehlerwirkung durchliefen.
PWF (passed, was failed)	Anzahl aktuell bestandener Testfälle, die vor der Änderung eine Fehlerwirkung aufdeckten.

Tabelle 6.5 Testfallbasierte Metriken (Spillner, Roßner, Winter, & Linz, 2014, S. 206)

TESTOBJEKTBASIERTE METRIKEN

Zusätzlich zu testbasisorientierten Metriken sind produktorientierte Testabdeckungsmaße erforderlich. Diese messen den Testfortschritt gegenüber dem Umfang der Testbasis. Zur Ermittlung der Testabdeckungsmaße wird die Abstraktionsebene der jeweiligen Teststufe verwendet (Spillner, Roßner, Winter, & Linz, 2014, S. 208).

Im System- und Abnahmetest können funktions- und anforderungsorientierte Metriken erhoben werden (Spillner, Roßner, Winter, & Linz, 2014, S. 208):

- Prozentsatz durch Testfälle abgedeckter Anforderungen
- Prozentsatz ausgeführter Ablaufpfade der Anwendungsfälle
- Prozentsatz fachlicher Datentypen, von denen Objekte in der Datenbank instanziiert / gelesen / geändert / gelöscht wurden

Im Integrationstest werden die einzelnen Architekturkomponenten und ihr Zusammenspiel über die Datenflüsse betrachtet.

- Prozentsatz getesteter Schnittstellen
- Prozentsatz getesteter Schnittstellenanwendungen
- Prozentsatz der mit Testentwurfsverfahren getesteten Äquivalenzklassen der Schnittstellenparameter

Bei Komponententests wird sich auf strukturorientierte Metriken fokussiert.

- Anzahl neuer bzw. geänderter Operationen
- Anzahl Codezeilen
- Codekomplexität
- Anzahl der Aufrufe von Funktionen im Kontrollflussgraphen

Die Komplexitätsmetriken können als Indikatoren für das Produktrisiko dienen. Wurden die Produktrisiken auf die Testobjekte abgebildet, lassen sich aus deren Überdeckung weitere Aussagen zur Risikoüberdeckung ableiten (Stauffer, Honegger, & Gisin, 2013, S. 210).

- Anzahl Testfälle pro Risiko
- Prozentsatz durch Testfälle überdeckter Risiken
- Prozentsatz bestandener/ nicht bestandener Testfälle pro Risiko

Weitere risikobasierte Metriken lassen sich aus dem Risikoinventar ableiten.

- Anzahl der verbleibenden Risiken nach Risikohöhe
- Anzahl der bewältigten Risiken nach Risikohöhe
- Aktuelle Tendenzen zu Risikoeintrittsindikatoren

Die Metriken fließen in die Testüberwachung und Teststeuerung ein. Anhand der Metriken ist der Testfortschritt zu bewerten. Die Informationen zum Testfortschritt, zur Produktqualität und zu den ergriffenen Steuerungsmaßnahmen müssen im Sinne einer hohen Transparenz regelmäßig an alle Stakeholder kommuniziert werden.

7 FAZIT UND AUSBLICK

Die Fragestellung, wie sichergestellt werden kann, dass ein BI-System bei iterativer Erweiterung weiterhin funktioniert, wurde aus der Sicht der drei Einflussdimensionen BI Agilität, BI-Architektur und Softwarequalität betrachtet.

Die Testpunkte eines BI-Systems ergeben sich aus der Architektur. Die wesentlichen Herausforderungen der BI-Architektur sind die heterogene Systemlandschaft und die Datenzentriertheit. Deshalb teilen die Autoren das System in kleinere Subsysteme auf. Diese homogenen Systemteile können dann entsprechend ihrer spezifischen Anforderung getestet werden. Die Integration erfolgt anhand der Datenflüsse. Die Autoren haben unterschiedliche Herangehensweisen bei der Zuordnung von Qualitätszielen und Integrationsstufen zu den Architekturebenen gewählt. Während bei (Rizzi & Golfarelli, 2009) der Architekturebene jeweils eine Integrationsebene und ausgewählte Qualitätsziele zugeordnet werden, bilden (Trahasch & Zimmer, 2016) einen Testwürfel, bestehend aus den Dimensionen Architekturebene, Qualitätsmerkmal und Integrationsstufe. Somit wird jede Architekturebene auf jeder Integrationsstufe gegen jedes Qualitätsmerkmal getestet.

Das Ergebnis dieser Arbeit kombiniert diese Herangehensweisen. Das BI-System wird anhand der Architektur in die Subsysteme Datenflüsse (ETL), Data Warehouse (DWH), Datamart und Frontend aufgeteilt. Die Komponenten werden unabhängig voneinander auf Funktionalität und Benutzbarkeit getestet. Das Testen auf Integrationsebene wird in technische und fachliche Testfälle aufgeteilt. Dies ist eine Konsequenz des iterativen Vorgehens. Für die technische Überprüfung reicht ein kleiner, methodisch erstellter Testdatensatz aus – ein Regressionstest kann täglich ausgeführt werden. Die fachliche Überprüfung erfordert umfassendere Daten. Deshalb wird der fachliche Regressionstest einmal wöchentlich ausgeführt. Der fachliche Test erfolgt, wenn möglich, anhand der aggregierten Daten auf Datamart Ebene. Oberflächentests haben eine längere Laufzeit und sind schwerer zu automatisieren. Tests auf Systemebene erfolgen aufgrund ihrer Laufzeit außerhalb der Iteration. Die Qualitätsmerkmale Sicherheit, Robustheit und Effizienz werden auf Systemebene getestet. Das Ergebnis ist ein Prozess, der die Aufgaben der Qualitätssicherung anhand der Teststufen in eine Iteration einplant. Der Prozess berücksichtigt explizit die Integration von automatisierten Regressionstests. *Explorative Tests ergänzen die Teststufen und tragen* dazu bei, mögliche Fehler und systematische Schwächen der methodisch aufgebauten Teststrategie zu vermeiden. Der Prozess wird um Techniken zur Testfallerstellung und Auswahl ergänzt.

Insbesondere auf Komponentenebene stellen die Autoren umfassende Listen mit möglichen Testfällen vor (Hughes, 2016). Diese sind sehr ausgereift und als Standardmethoden im BI-Umfeld anerkannt. Deshalb wurden diese im Rahmen dieser Arbeit nicht weiter betrachtet, sondern nur Auszüge möglicher Testfälle für einzelne Qualitätskriterien angeführt.

Der erste Teil der Ausgangsfrage bezieht sich auf „das Sicherstellen“ der Funktionalität. Es ist also notwendig, anhand der Testfälle die Funktion des Systems zu quantifizieren. Hierzu liefert die Literatur zum BI spezifischem Testen nur wenig Anhaltspunkte. Lediglich bei (Rizzi & Golfarelli, 2009) findet sich eine Methodik geeignete Metriken festzulegen. Deshalb wurden in dieser Arbeit die BI-spezifischen Methoden um quantifizierbare Methoden aus der klassischen Software Entwicklung ergänzt. Diese können in Methoden zum Testentwurf und Methoden zur Qualitätsmessung unterteilt werden. Insbesondere die Äquivalenzklassenbildung – kombiniert mit einer Grenzwertanalyse – eignet sich zum Prüfen der Funktionalität in allen Integrationsstufen. Zustandsbezogene Testfälle eignen sich für die ETL Prozesse. Neben dem Eingabewert, hat auch der bisherige Ablauf des Systems Einfluss auf das Systemverhalten. Entsprechend kann auch die Testabdeckung auf Integrationsebene anhand der durchlaufenen ETL Prozesse gemessen werden.

Um die Produktqualität zu quantifizieren, kommen testfallbasierte und testobjektbasierte Metriken zum Einsatz. Somit kann projektindividuell festgelegt werden, mit welcher Wahrscheinlichkeit sichergestellt sein muss, dass das Produkt nach einer Veränderung weiterhin funktioniert. Es gilt jedoch festzuhalten, dass „selbst wenn alle ausgeführten Testfälle keinen einzigen Fehler mehr aufdecken, kann nicht mit völliger Sicherheit ausgeschlossen werden, dass es nicht zusätzliche Testfälle gibt, die weitere Fehlerwirkungen erzeugen und somit weitere Fehlerzustände in der Software aufzeigen.“ (Spillner & Linz, 2010, S. 10). Durch das Testen wird jedoch das Risiko beim Einsatz der Software verringert. Dieses Risiko lässt sich nur anhand von Metriken quantifizieren.

Zunächst gilt es, den praktischen Nutzen der Strategie zu verifizieren. Dies kann aus verschiedenen Perspektiven geschehen. Die Nutzbarkeit kann anhand der beiden Perspektiven „Eingliederung in den Prozess“ und „Umsetzbarkeit des Testfallentwurfs“ erfolgen. Die Rückmeldung der Prozess Stakeholder ist hier essenziell. Die Reife des Prozesses lässt sich anhand der erhobenen Qualitätsmetriken messen. Hierzu ist es jedoch notwendig, dass diese bereits vorher erhoben wurden. Metriken können Grundlage für die Optimierung der Strategie sein.

Aus den Einflussdimensionen Agilität und Softwarequalität ergeben sich weiterführende Fragestellungen im Hinblick auf das iterative Testen von BI-Systemen. Das Themengebiet risikoorientiertes Testen wurde sowohl in dieser Arbeit, aber auch bei anderen Autoren nur oberflächlich behandelt. Die risikobasierte Testauswahl ist insbesondere bei zeitlicher Restriktion ein sinnvolles Mittel die Menge an Testfällen zu reduzieren. Hier gilt es zu prüfen, welche spezifischen Risiken aus der BI-Architektur entstehen und wie diese bewertet und behandelt werden können. Des Weiteren bleibt die Frage nach optimalen Testdaten offen. Was ist die optimale Größe der Datensätze und mit welcher Methodik werden diese erstellt? Die Komplexität des Themas entsteht aus der Kombination der einzelnen Themengebiete. Grundsätzlich gilt es, die Thematik in handhabbare Teile aufzuschlüsseln und sich der Frage iterativ zu nähern.

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LITERATURVERZEICHNIS

Alpar, P., Alt, R., Bensberg, F., Grob, H. L., Peter, W., & Robert, W. (2014). *Anwendungsorientierte Wirtschaftsinformatik* (Bd. 7). Wiesbaden: Springer Vieweg. doi:10.1007/978-3-658-00521-4

Bange, C. (2006). *Analytische Informationssysteme. Business Intelligence-Technolo-gien und -Anwendungen. Werkzeuge für analytische Informationssysteme.* Heidelberg: Springer.

Bauer, A., & Günzel, H. (. (2013). *Data-Warehouse-Systeme. Architek-tur, Entwicklung, Anwendung.* Heidelberg: dpunkt.verlag.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Sutherland, J. (2001). *Principles behind the Agile Manifesto*. (The Agile Alliance, Produzent) Abgerufen am 27. 11 2017 von http://agilemanifesto.org/principles.html

Bucsics, T., Baumgartner, M., Seidl, R., & Gwihs, S. (2015). *Basiswissen Testautomatisierung* (2nd ed. Ausg.). Heidelberg: dpunkt.verlag.

Cohn, M. (2009). *Succeeding with Agile: Software Development Using.* Amsterdam: Addison-Wesley Professional.

Collier, K. (2012). *Agile analytics.* Upper Saddle River, NJ: Addison-Wesley.

Duvall, P. M., Matyas, S., & Glover, A. (2013). *Continuous integration* (8. print Ausg.). Upper Saddle River, NJ: Addison-Wesley.

Gabriel, R., Gluchowski, P., & Pastwa, A. (2009). *Data warehouse & Data Mining.* Herdecke, Witten: W3L-Verl. (Informatik).

Göhl, & Hahne. (2 2012). Bessere Architektur, Organisation und Methodik. *BI-Spektrum*, S. 10-15.

Hahne, M. (2014). *Modellierung von Business-Intelligence-Systemen.* Heidelberg: dpunkt.verlag GmbH.

Hansen, H. R., & Neumann, G. (2009). *Wirtschaftsinformatik 1 - Grundagen und Anwendungen.* Tübingen: Lucius & Lucius Verlagsgesellschaft mbH.

Hughes, R. (2016). *Agile data warehousing for the enterprise.* Waltham, MA: Morgan Kaufmann.

ISO/IEC. (06 2001). 9126-1. ISO/IEC JTC 1/SC 7 Software and systems engineering.

ISO/IEC. (2008). ISO/IEC 27005:2008. International Organization for Standardization.

ISO/IEC. (09 2015). ISO 9000:2015. ISO/TC.

ISTQB. (21. 05 2017). *ISTQB GTB Standardglossar der Testbegriffe*. (G. T. ISTQB AISBL, Herausgeber) Abgerufen am 2. 12 2017 von ISTQB GTB Standardglossar der Testbegriffe: http://glossar.german-testing-board.info/

Kemper, H. G., & Finger, R. (2006). *Analytische Informationssysteme. Business Intelli-gence-Technologien und -Anwendungen. Transformation operativer Daten.* Heidelberg: Springer.

Kemper, H. G., Baars, H., & Mehanna, W. (2010). *Business Intelligence – Grundlagen und praktische Anwendung. Eine Einführung in die IT-basierte Managementunterstützung* (Bd. 3. überarb. Aufl.). Wiesbaden: Vieweg,.

Krawatzeck, R., Zimmer, M., & Trahasch, S. (2013). Agile Business Intelligence — Definition, Maßnahmen und Herausforderungen. *HMD Praxis der Wirtschaftsinformatik, 50*(2), S. 56–63.

Linz, T. (2017). *Testen in Scrum-Projekten. Leitfaden für Softwarequalität in der agilen Welt* (2nd ed. Ausg.). Heidelberg: dpunkt.verlag.

Neveen ElGamal. (2013). Data warehouse testing. In G. Guerrini, & K.-U. Sattler, *Proceedings of the Joint EDBT/ICDT 2013 Workshops* (S. 1–8). New York, NY: ACM.

Rizzi, S., & Golfarelli, M. (2009). A comprehensive approach to data warehouse testing. In I.-Y. Song. New York, NY: ACM.

Sam Schutte, Thilini Ariyachandra, & Mark Frolick. (2013). Test-Driven Development of Data Warehouses. In R. T. Herschel, *Principles and applications of business intelligence research.* Hershey, Pa: IGI Global (701 E. Chocolate Avenue Hershey Pennsylvania 17033 USA).

Schwaber, K., & Sutherland, J. (Juli 2013). *Der Scrum Guide.* Abgerufen am 09. 02 2018 von https://www.scrum.org: https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-DE.pdf

Spillner, A. (kein Datum).

Spillner, A., & Linz, T. (2010). *Basiswissen Softwaretest* (4. Aufl. Ausg.). s.l.: dpunkt.verlag.

Spillner, A., Roßner, T., Winter, M., & Linz, T. (2014). *Praxiswissen Softwaretest Tetsmanagement.* Heidelberg: dpunkt.verlag.

Stauffer, H., Honegger, B., & Gisin, H. (2013). *Testen von Data-Warehouse- und Business-Intelligence-Systemen (Edition TDWI)* (1. Aufl. Ausg.). s.l.: dpunkt.

Trahasch, S., & Zimmer, M. (Hrsg.). (2016). *Agile Business Intelligence* (1. Auflage Ausg.). Heidelberg: dpunkt.verlag.

Winter, M., Roßner, T., Brandes, C., & Goetz, H. (2016). *Basiswissen modellbasierter Test* (2., vollständig überarbeitete und aktualisierte Auflage Ausg.). Heidelberg: dpunkt.verlag.

Zimmer, M. (2015). *Agile Business Intelligence - Komponenten integrierter Gesamtarchitekturen* (Bd. 83). Köln: EUL Verlag.