Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Institut für Theoretische und Technische Informatik

Fachgebiet Softwarearchitekturen und Produktlinien

Dissertation

„Entwicklung und Evaluierung einer Erweiterung der BPMN-Spezifikation für einen modellbasierten und automatisierten Regressionstest verteilter BIS“

zur Erlangung der Würde

eines Doktor-Ingenieurs (Dr.-Ing.)

vorgelegt von

Khalid Ebanhesaten

1) Gutachter: Prof. Dr. Detlef Streitferdt JP (Technische Universität Ilmenau)
2) Gutachter: Prof. Dr. Dirk Stelzer (Technische Universität Ilmenau)
3) Gutachter: Prof. Dr. Elke Pulvermüller (Universität Osnabrück)

Tag der Einreichung:	16.07.2014
Tag des Rigorosums:	03.12.2014
Tag der wissenschaftlichen Aussprache:	03.12.2014

urn:nbn:de:gbv:ilm1-2014000420

Danksagung

Nach vielen Jahren intensiver Forschung neben einer Vollzeittätigkeit ist es endlich vollbracht. Damit ist es an der Zeit, mich bei denjenigen zu bedanken, die mich in dieser spannenden Phase begleitet haben. Zum besonderen Dank bin ich Professor Detlef Streitferdt verpflichtet, der mich stets mit Anregungen unterstützt hat und mich in den vergangenen Jahren mit bereichernden Tipps und Diskussionsbeiträgen wiederholt in neue thematische Bahnen gelenkt hat. Auch Fr. Professor Elke Pulvermüller und Herrn Professor Dirk Stelzer bin ich für das zweite und dritte Gutachten zum Dank verpflichtet. Zum Schluss möchte ich meiner besonderen und großartigen Familie danken – ohne irgendeinen auszulassen der den Namen Ebanhesaten trägt! Natürlich auch vieler meiner Freunde die mich in dieser Zeit begleitet haben.

„*Wer auch nur eine Spur Überheblichkeit in seinem Herzen hat, wird nicht ins Paradies eintreten*“ Zitat des Propheten Muhammad s.a.s.

Abstract

The constantly growing complexity of hard- and software systems combined with increasing functional scope and steadily decreasing product lifecycles is leading to an increased usage of model based technology in development and testing. Many challenges come along with developing and testing such complex software systems. Top priority of each company is to deliver the required software solution in the agreed time and budget frame. But reality shows, that time and cost pressure is massively affecting the quality of the delivered software. Therefore new testing methods have to be introduced in an early stage of the development process. While model based development procedures have found their way into embedded software systems during the past few years, model based testing (MBT) of distributed business information systems (BIS) has come more and more in the focus of research and industry. This dissertation provides an overview of the state of the art in MBT within the domain of distributed BIS and summarizes the actual research questions. Based on these research questions the requirements to such a test approach arose, which are being answered within the implementation section of this dissertation. As a modelling notation the Business Process Modelling Notation (BPMN) is being used, differentiating this MBT approach from other approaches out of embedded domain, which are using UML. Based on this and with the example of a debitor process (IT supported payment process) from the business-to-business (B2B) it will be shown, how an MBT approach can contribute to a sustainably quality increase as improved well as test coverage of distributed BIS.

Therefore within the frame of the dissertation testing tools developed by Seppmed GmbH and MID GmbH are being evaluated and analyzed on real world usage capabilities. In detail it is being investigated, how far the testing tools in combination with other applications are fulfilling the requirements of the dissertation. Thus, the proposed methodology was applied in an industrial setting. A cost analysis as well as impacts of MBT implementation in existing development processes will be provided in the last part of this thesis.

Kurzfassung

Die stetig wachsende Komplexität von Hard- und Softwaresystemen sowie der steigende Funktionsumfang bei stets kürzer werdenden Produktzyklen führen zunehmend zum Einsatz von modellbasierten Technologien in der Entwicklung und bei den Tests. Bei der Entwicklung und beim Test solcher komplexer Softwaresysteme werden Unternehmen heute mit neuen Herausforderungen konfrontiert. Oberste Maxime eines jeden Unternehmens ist, dass die angeforderte Softwarelösung im vereinbarten Zeit- und Budgetrahmen geliefert werden muss. Die Praxis zeigt allerdings, dass sich Zeit- und Kostendruck massiv auf die Qualität der ausgelieferten Software auswirken. Neue Testmethoden müssen daher in einem frühen Stadium des Softwareentwicklungsprozesses Einzug halten. Nachdem modellbasierte Entwicklungsmethoden bereits in den letzten Jahren zunehmend in eingebetteten Softwaresystemen Einzug in die Entwicklungsabteilungen gehalten haben, ist das modellbasierte Testen (MBT) von verteilten betrieblichen Informationssystemen (BIS) verstärkt in den Fokus von Forschung und Industrie gerückt. Diese Dissertation gibt einen Überblick über den Stand der Technik bezüglich des MBT in der Domäne der BIS und fasst aktuelle Forschungsfragen zusammen. Aus den Forschungsfragen lassen sich Anforderungen an einen solchen Testansatz formulieren, die dann im technischen Realisierungsteil dieser Dissertation beantwortet werden. Anders als in den bisherigen MBT-Ansätzen aus der Domäne der eingebetteten Systeme wird als Modellierungsnotation nicht auf UML, sondern auf Business Process Modeling Notation (BPMN) gesetzt. Basierend darauf wird anhand eines Fallbeispiels mit einem betriebswirtschaftlichen Debitorenprozess (IT-gestützter Zahlungsabwicklungsprozess) aus dem Business-to-Business (B2B) gezeigt, wie ein MBT-Ansatz einen Beitrag dazu leisten kann, sowohl die Testabdeckung als auch die Testqualität von verteilten BIS nachhaltig zu verbessern. Dazu werden im Rahmen dieser Dissertation Testwerkzeuge der Firmen Seppmed GmbH und der MID GmbH evaluiert und auf ihre Praxistauglichkeit untersucht. Dabei soll im Speziellen untersucht werden, inwiefern diese Werkzeuge in Kombination mit Testwerkzeugen die Anforderungen dieser Dissertation erfüllen. Für die Modellierung der Testfälle in BPMN ist es notwendig, die BPMN-Spezifikation um Testspezifika zu erweitern. Die so gewonnenen manuellen Testfälle werden durch eine spezielle Adaptierung, welche im Rahmen dieser Dissertation entwickelt wurde, in fertig automatisierte und ausführbereite Testskripte überführt. So entsteht eine Methode, welche es ermöglicht, Geschäftsprozesse, die verteilt über mehrere verschiedene Softwaresysteme abgewickelt werden, End-to-End zu testen. Da die Notwendigkeit und die Motivation dieser Dissertation aus der Praxis entstanden, wird die entwickelte Methode dementsprechend unter Praxisbedingungen erprobt und evaluiert. Dabei konnte festgestellt werden, dass sowohl die Testabdeckung (Prozessabdeckung) als auch die Testqualität am Fallbeispiel erhöht werden konnten und gleichzeitig eine Testzeitersparnis von ca. 50% erreicht wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ALM	Application Lifecycle Management
BIS	Betriebliche Informationssysteme
BPMN	Business Process Modeling Notation
BPMI	Business Process Management Initiative
CFM	Customer File Managementsystem
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
HPQC	Hewlett Packard Quality Center
IBA	Innovator for Business Analyst
IT	Informationstechnologie
Jar	Java Archivdateien
MBTSuite	Model Based Testing Suite
MBT	Modellbasiertes Testen
OMG	Object Management Group
QTP	Quicktest Professional
ROI	Return on Investment
SAP-FI	Systeme – Anwendungen – Programme, Modul Finanzen
SAP-FICO	SAP Controlling
SDI	Serial Document Interface
SUT	System Under Test
UML	Unified Modeling Language
VBA	Visual Basic
XSD	Exentsible Schema Definition
XT	Extreme Tailoring

1 Motivation

Hersteller von Softwaresystemen für verteilte betriebliche Informationssysteme (BIS) sind einem enormen Wettbewerbsdruck ausgesetzt und müssen den hohen Anforderungen am Markt genügen, um konkurrenzfähig zu sein. Der Wettbewerbsdruck wird durch den Kunden erzeugt, indem in immer kürzer werdenden Zyklen Produkte erwartet werden, bei gleichzeitiger Qualitätsverbesserung. Durch den Softwaretest soll stichprobenartig und risikobasiert die Konsistenz zwischen Anforderungsspezifikation und Implementierung geprüft werden [1]. Risikobasiertes Testen bedeutet, dass aufgrund von Ressourcen- und Zeitengpässen nur die wichtigsten und kritischsten Softwarepfade getestet werden können. Testen zählt zu den wichtigsten qualitätssichernden Maßnahmen in der Softwareentwicklung [1]. In der Domäne der betrieblichen Informationssysteme dominieren bislang dokumentenbasierte Methoden den Testprozess [2]. Dokumentenbasierte Methoden können allerdings den heutzutage steigenden Anforderungen an ein Softwaresystem, wie immer kleiner werdende Releasezyklen, hohe Wartbar- und Wiederverwendbarkeit bei gleichzeitig kürzeren Testzyklen, kaum erfüllen [2]. Aus diesen Anforderungen an ein Softwaresystem resultieren immer komplexere und durch Schnittstellen vernetzte Softwaresysteme mit ständig steigender Funktionalität bei gleichzeitiger Verkürzung der Produktzyklen. Werden weiterhin die klassischen[1], dokumentbasierten[2] Testprozesse eingesetzt, sinken die Qualität und die Transparenz solcher Softwaresysteme nachhaltig, da eine hohe Testabdeckung kaum erreicht werden kann oder nur mit einem unvertretbar hohen Aufwand zu erreichen wäre. Mit Testabdeckung ist die Abdeckung gemeint, welche sich auf Geschäftsprozesse bezieht, so können nicht alle Geschäftsprozesse während einer Testphase getestet werden. Eine Testautomation im klassischen Sinne (Capture & Replay) einzuführen, kann bei solchen komplex vernetzten Softwaresystemen dazu führen, dass die Kosten den Nutzen überragen [2], [3]. So kann es vorkommen, dass Testfälle automatisiert werden, die vorher nicht auf Automatisierbarkeit untersucht worden sind [77]. Das Ergebnis kann sein, dass ein Testfall mit sehr viel Aufwand automatisiert (programmiert) wird, dieser allerdings bei jedem Testdurchlauf angepasst werden muss, so dass das entwickelte Testskript einem ständigen Anpassungsprozess unterliegt und letztendlich mehr Kosten als Nutzen bringt [4]. Um dem hohen Wettbewerbsdruck trotzdem entgegenzutreten, muss eine Testmethode entwickelt werden, die das Testen von verteilten BIS auch mit geringerem Zeit- und Ressourcenaufwand ermöglicht. Dabei ist es das Ziel, die Testabdeckung so weit zu erhöhen, dass eine Aussagekraft über die Qualität eines Softwaresystems ermöglicht wird. Eine neue Testmethode in der Domäne verteilter komplexer BIS beschreibt das modellbasierte Testen (MBT) [88]. Dabei implementiert das MBT nicht nur eine neue Testmethode, sondern soll den bisherigen Testprozess, wie er in [5] oder auch [8] beschrieben ist, erweitern.

[1] Unter den klassischen Testprozessen werden die Testprozesse verstanden, welche sich allein an der in Textform vorliegenden funktionalen Spezifikation orientieren und auf Basis dieser Testfälle in Textform dazu definieren, ohne den Einsatz von Requirement-Engineering-Werkzeugen oder Modellierungswerkzeugen zur Definition von Testprozessen.

[2] Darunter werden die aus Textdokumenten erstellten Testfälle verstanden.

Von der Mischung aus einer skriptbasierten Testautomation mit Testmodellen sollen bisherige klassische Testverfahren sukzessive abgelöst werden, wobei hier die Fokussierung auf den Regressionstestfällen liegt.

1.1 Problembeschreibung

Mit dem Boom des Internets und der hohen Verbreitung eingebetteter Systeme ist in den letzten Jahren Software in viele sicherheitskritische Bereiche des täglichen Lebens eingedrungen [2]. Bei der Entwicklung und Qualitätssicherung von solch komplexen und sicheren Softwaresystemen werden Unternehmen heute mit ganz neuen Herausforderungen konfrontiert. Neben anderen konstruktiven und analytischen Maßnahmen spielt vor allem das dynamische Testen eine herausragende Rolle zur Absicherung der Qualität einmal erstellter Software [6], [7]. Abgesehen von grundsätzlichen Einwänden der Art, dass durch das Testen weder Qualität erzeugt noch die Abwesenheit von Fehlern gezeigt werden kann, gibt es jedoch auch heute noch viele offene Probleme, etwa bei der Automatisierung von Softwaretests und bei der Bestimmung effektiver Testüberdeckungsmaße [8].

Durch den Softwaretest soll stichprobenartig und risikobasiert die Konsistenz zwischen Anforderungsspezifikation und Implementierung geprüft werden. Da eine manuelle Ableitung der Testfälle aus den Anforderungen aufwendig und fehlerträchtig ist und oft intuitiv und unsystematisch geschieht, ist eine systematisch automatisierte Testfallerstellung wünschenswert [9]. Dadurch ist in den letzten Jahren zunehmend das modellbasierte Testen (MBT) in den Fokus von Forschung und Industrie gerückt [80]. Ausgehend von einer funktionalen Spezifikation der Anforderungen in natürlicher Sprache wird dabei ein plattformunabhängiges Testmodell erstellt, welches schrittweise erweitert werden kann. Die funktionale Spezifikation enthält die präzise Formulierung der gewünschten Systemfunktionalität und die Übereinstimmung der Funktionalität des entwickelten mit dem erwünschten System. Dabei werden in diesem Rahmen nicht funktionale Anforderungen bewusst ausgeklammert und nicht behandelt. Systeme, von deren einwandfreiem Verhalten Menschenleben abhängen können (bspw. Airbags, Bremssysteme, Herz-Lungen-Maschinen), und solche, deren Fehlverhalten zu Imageschäden (Chipkarten) und kostspieligen Rückrufaktionen (Benzinpumpen in Automobilen) führen, illustrieren das eindringlich. Um sowohl die gewünschte Funktionsfähigkeit als auch eine gewisse Qualität zu erreichen, müssen die Anforderungen an ein System überprüfbar gestaltet werden. Mit „gewisse Qualität“ ist gemeint, dass alle Kriterien aus der Domäne der Softwarequalitätssicherung [5] erfüllt sein müssen. Allerdings erfordern solche „Systems of Systems“ einen umfangreichen Test aller angebundenen Systeme, was meistens aus Zeit- und Ressourcengründen nicht durchgeführt werden kann. Nach [1] steht das klassische Testen, was heute vorwiegend in der Domäne der BIS vorzufinden ist, vor denselben Problemen wie vor Jahren: die Systematik bei der Erhebung von Testfällen ist oftmals nicht nachvollziehbar, wodurch sich der Gesamtprozess nicht reproduzieren lässt. Ein vollständiger Test aller Softwareanteile ist mit der großen Anzahl dafür notwendiger Testfälle nicht realisierbar. Somit können nicht immer alle softwarekritischen Pfade beim Testen durchlaufen werden [10]. Der Begriff „Softwareanteile“ steht für alle Systeme, die dazu dienen, einen Geschäftsprozess erfolgreich auszuführen. Die einzelnen Teilsysteme sind dabei in einem hohen Maße voneinander abhängig. Deshalb müssen klassische Testansätze [5] durch weitere Testmethoden erweitert werden,

um das Testen von verteilten BIS mit einem hohen Automatisierungs- und Abdeckungsgrad zu ermöglichen.

Allerdings gilt die Regel [5], dass Tests nur die Existenz von Fehlern aufzeigen können, nicht die Korrektheit und die Fehlerfreiheit eines Softwaresystems. Aber auch diese Regel ist mittlerweile veraltet, da Tests heute zusätzlich zum Nachweis von Qualität genutzt werden. Gleichzeitig erhöht eine wenig fehlerbehaftete Software das Vertrauen in die Zuverlässigkeit einer Software. MBT ist eine Technik, die den Testprozess – insbesondere die Kernphasen (manuelles Testen, automatisiertes Testen) – durch den Einsatz von Testmodellen unterstützt, wobei nur das Modellieren der Testfälle manuell erfolgt. Das Generieren von Testfällen aus den Testmodellen sowie deren Ausführung sollen automatisiert erfolgen. Trotz intensiver Testbemühungen aller an einem betriebswirtschaftlichen Geschäftsprozess beteiligten Applikationen kommt es in den Produktivsystemen zu Fehlern, die durch umfangreichere, automatisierte Regressionstests gefunden werden könnten. Mit MBT, was für verteilte BIS auch heute noch eine neue Testmethodik darstellt, sind Hoffnungen an eine höhere Testabdeckung solcher Applikationen verknüpft. Eine hohe Testabdeckung garantiert keine Fehlerfreiheit, sie kann aber dazu führen, sukzessive die Qualität von Softwaresystemen zu steigern.

1.2 Wissenschaftliche Motivation für MBT in verteilten BIS auf Basis von BPMN

In erster Linie werden von dieser Methode die signifikante Erhöhung der Testqualität und die methodische Unterstützung der Testphasen für betriebliche Informationssysteme erwartet. Gleichzeitig soll ein wissenschaftlicher Beitrag in der Domäne des BPMN-basierten Testens verteilter BIS geleistet werden. Dieses Gebiet ist in bisherigen wissenschaftlichen Arbeiten und in der Forschung[3] selten zu finden [12], [13], [165], [166]. Die Ergebnisse dieser Dissertation können als Grundlage für zukünftige wissenschaftliche Arbeiten auf diesem Gebiet genommen werden. Entscheidende Argumente sind vor allem die Vereinfachung der Handhabung der Modellierung und die frühzeitige Einbindung von Fachbereichen in einer Testphase.

Bestehende Ansätze zur Testfallermittlung mittels BPMN-Modellen [12], [13] sind unzureichend. Auch liegen entweder keine Dokumentationen einer praktischen Erprobung vor oder sie erweisen sich bei einer detaillierten Untersuchung für die Praxisanwendung als unzureichend. Der Hintergrund, warum ausgerechnet die BPMN hier ausgewählt wird, ist einfach: die BPMN ist eine leicht zu erlernende Notation, welche bereits in mehreren Unternehmen als Modellierungsnotation von Geschäftsprozessen genutzt wird [151]. Genau auf diesem Wissen kann jetzt aufgebaut werden, um das Testen auf eine abstraktere Ebene zu bringen. Die BPMN in der Standardausgabe kann allerdings nicht für die Testfallableitung genutzt werden, es bedarf einer Erweiterung der Notation um Testspezifika und Elemente. Daher wird die automatisierte Testfallableitung aus BPMN-Testmodellen über eine Erweiterung der BPMN um Testelemente (Testnotation) realisiert.

[3] Recherche in Web of Science, Paper, Literatur.

Durch die Abstraktion der Tests auf BPMN-Testmodelle können die Testqualität und die Prozessabdeckung nachhaltig erhöht werden. Zusätzlich können durch die Einbindung von Fachbereichen bei der Testfallmodellierung frühzeitig Fehler und Widersprüche in funktionalen Anforderungen identifiziert und behoben werden. Es findet ein Mix aus der konstruktiven und der analytischen Qualitätssicherung statt, denn genau diese Kombination kann dazu beitragen, Fehler frühzeitig zu erkennen und gleichzeitig zu beheben. Daraus resultiert, dass Fehlerkosten in Produktivumgebungen deutlich gesenkt werden können. Demgegenüber stehen hohe Initialaufwände beim Modellieren und bei der Programmierung der automatisierten Testskripte. Ein weiterer Vorteil einer Testnotation ist die unmissverständliche Interpretierbarkeit der Elemente durch einen Testfallgenerator. Hierbei entstehen Testmodelle in unterschiedlicher Abstraktionstiefe, von denen abhängig von der auszuwählenden Generierungsstrategie (Pfad-, Kanten-, Knotenüberdeckung) Testfälle abgeleitet werden können. Durch den Einsatz von Testmodellen bei der Modellierung von Testfällen kann gleichzeitig eine Prozessabdeckung als Kennzahl abgeleitet werden, die sich konkret aus der Summe der unternehmenskritischen Geschäftsprozesse errechnet. Die Kennzahl zeigt an, welcher Testabdeckungsgrad durch die Tests zu erwarten ist.

1.3 Zielsetzung

In einem Satz zusammengefasst, ist das Ziel der Arbeit die Entwicklung und die Evaluierung einer Erweiterung der BPMN-Spezifikation für den modellbasierten und automatisierten Regressionstest verteilter BIS. Es wird ein Set an regressiven Testskripten benötigt, welches sukzessive um Tests erweitert wird. Die Methode soll bisherige wissenschaftliche Ansätze zur BPMN und zum Testen [12], [13], [165], [166] grundlegend erweitern und somit einen wissenschaftlichen Beitrag leisten. Gleichzeitig wird die Entwicklung dieser Methode unter Praxisbedingungen evaluiert und auf Praxistauglichkeit untersucht, wobei nicht nur Testfälle aus Testmodellen generiert werden, sondern in mehreren noch zu entwickelnden Adaptierungen fertig automatisierte Regressionstestskripte (Regressionstestfälle) entstehen. Dadurch soll ein End-to-End-Regressionstest von Geschäftsprozessen verteilter BIS ermöglicht werden. Auf der Basis der BPMN-Notation, welche schrittweise Einzug in Unternehmensorganisationen (Fachbereiche) erhält [151], [152], werden Testmodelle in Zusammenarbeit mit IT-Organisationen entworfen und fachlich modelliert. Damit wird ein weiteres Ziel verfolgt, nämlich die Testmodelle gleichzeitig durch Fachbereiche auf Widersprüche, Vollständigkeit und Konsistenz überprüfen zu lassen. Dabei steht auch das pragmatische Wissenschaftsziel dieser Dissertation im Vordergrund, sprich, wissenschaftlich gewonnene Erkenntnisse müssen demzufolge zur Lösung praktischer Probleme (also in der Praxis) nutzbar sein. Dem pragmatischen Wissenschaftsziel folgend ist diese Arbeit an den Bedürfnissen der Praxis ausgerichtet. Daher basieren die im Rahmen dieser Dissertation getroffenen Annahmen auf praktischen Gegebenheiten. Die hier durchzuführende Kosten-Nutzen-Analyse kann erst einmal nur die Kosten sowie den Nutzen, gemessen an dem erzielten Testabdeckungsgrad und an der Beschleunigung der Testausführung, gegenüberstellen.

1.4 Aufgabenstellung und Evaluierungskriterien

Verteilte BIS haben sich zu unternehmensweiten Informationssystemen entwickelt, die das Rückgrat der gesamten Informationsverarbeitung in Unternehmen darstellen. Unter diesem Aspekt gewinnen die Themen des modellbasierten Testens betriebswirtschaftlicher Informationssysteme zunehmend an Bedeutung. Unternehmen investieren seit Jahren erhebliche Summen, um Wettbewerbsvorteile, wie beispielsweise schnellere Reaktionszeiten auf Kundenanforderungen oder effizientere Planungsstrategien, auf der Grundlage von BIS zu schaffen [11]. Die Probleme, zu deren Lösung diese Arbeit Beiträge in technologischer und methodischer Hinsicht leistet, lassen sich einfach an folgender Problembeschreibung verdeutlichen, welche zeigen soll, wie kompliziert mit klassischen Testmethoden in verteilten Applikationen (Softwaresystemen) eine hohe Testabdeckung[4] zu erreichen ist. Der debitorische Geschäftsprozess (IT-gestützter Zahlungsabwicklungsprozess) ist ein unscheinbarer, einfacher Prozess, an dem viele betriebswirtschaftliche Applikationen beteiligt sind. Funktioniert an einer Stelle im Prozess eine Applikation nicht, wie die Spezifikation und der Geschäftsprozess es vorgeben, kann das zu Fehlern führen, die einem Unternehmen nicht nur schwer bezifferbare Imageschäden, sondern auch hohe Kosten verursachen können.

Ein Versuch, diese verteilten Applikationen mit herkömmlichen Testmethoden und Testprozessen [8] zu testen, ist aufgrund der Komplexität und des Umfangs nur mit risikobasierten Tests in einer vertretbaren Zeit möglich. Risikobasiert bedeutet, dass zum Test nur eine Auswahl von geschäftskritischen Prozessen erfolgt. Dabei kann es passieren, dass Fehler unentdeckt bleiben und so in die Produktionssysteme gelangen. Wichtige Pfade können beim risikobasierten Test (aber auch generell) übersehen oder vergessen werden. Weiter ist auf der Basis geschäftskritischer Testfälle eine Schätzung der Testabdeckung nur schwer oder gar nicht möglich. Zusätzlich kommt erschwerend hinzu, dass die verteilten Applikationen zumeist einem nicht einheitlichen Releasezyklus unterliegen, so dass diese nicht durchgängig integrativ getestet werden können. Zu jedem Release muss die Applikation sowohl integrativ (bezüglich der neu umgesetzten Funktionalitäten) als auch regressiv (alle vorherigen Funktionalitäten müssen noch vorhanden sein) getestet werden. Dies erfolgt im ersten Schritt losgelöst von allen anderen angebundenen Systemen, anschließend erfolgt der integrative Test über alle angebundenen Systeme. Unabhängig davon erfordern auch die angebundenen Systeme bei jeder Veränderung in den Systemen einen applikationsübergreifenden Regressionstest. Der Gefahr der Testfallexplosion entgeht man durch den Einsatz von Risikoklassen und Priorisierungen in der Testfallerstellung. Die klassische Testautomatisierung, welche auf Capture & Replay und nur in wenigen Fällen auf der aufwendigen deskriptiven Programmierung basiert, unterstützt nur bedingt bei der Beschleunigung der Testausführung [77], [119].

Die klassische Testautomatisierung ist sinnvoll, um die Zeit, die zur Ausführung der Tests benötigt wird, zu minimieren, Kosten zu sparen und die Qualität der Tests (durch Variantenreichtum) zu erhöhen. Ein weiterer nützlicher Einsatz der Automatisierung ist die häufige Wiederholung eintöniger Testprozeduren [9]. Die Praxis zeigt allerdings auch, dass

[4] Mit der Testabdeckung lässt sich messen, welche Bereiche des Programmcodes eines Systems beim Ausführen von Tests durchlaufen werden und welche nicht.

Testskripte bei jeder kleinen Veränderung im System angepasst werden müssen. Solche Anpassungen können im schlimmsten Falle dazu führen, dass Testskripte komplett neu programmiert werden müssen. Weiter zeigt die Praxis auch, dass die klassischen Testautomationsansätze (für fachliche Blackbox-Tests) [9], [10] meist dort eingesetzt werden, wo Tester durch Routinetätigkeiten aufgehalten werden. Ein automatisierter Testprozess ist sicherlich wünschenswert, aber wegen der speziell für BIS verteilten Eigenschaften in der heutigen Form nicht ausreichend. Die Forschungsfragen (Anforderungen) an diese Dissertation lassen sich wie folgt formulieren:

1. Welche Ansätze gibt es bereits beim MBT und in der Domäne eingebetteter Systeme und wie können diese auf die Domäne betrieblicher Informationssysteme übertragen werden?

2. Welche Testmethoden existieren und wo liegen deren Grenzen in Bezug auf verteilte BIS?

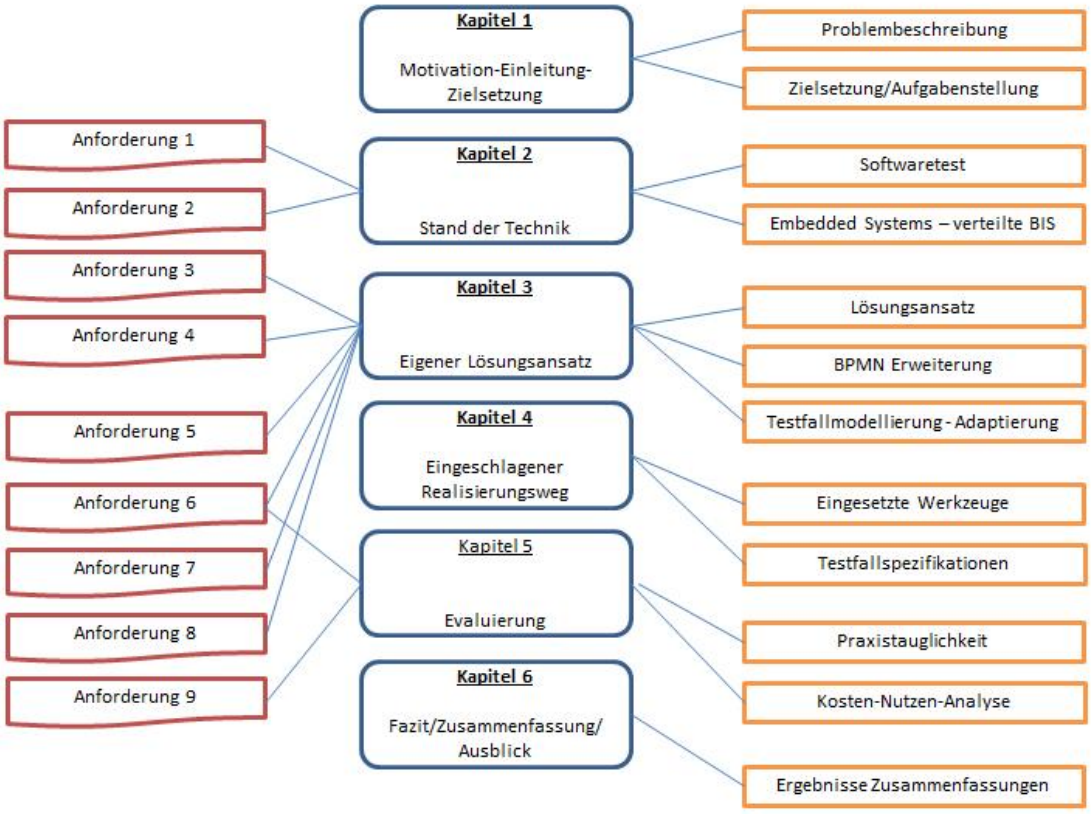


Abbildung 1: Anforderungen an die Dissertation und die Zuordnungen in Kapitel

3. Welche Testwerkzeuge existieren in der Domäne der BIS, wo werden diese eingesetzt, mit welchem Erfolg und warum ist eine Erweiterung hinsichtlich verteilter BIS notwendig? Hier wird eine Auswahl führender Testwerkzeuge aufgeführt.

4. Welche Modellierungsnotationen existieren im Zusammenhang mit MBT und wo liegen deren Grenzen?

5. Wie kann ein MBT-Testwerkzeug in ein etabliertes „Application Lifecycle Management (ALM)“-Tool integriert werden, um so eine Schnittstelle zu Requirement-Engineering-Werkzeugen zu ermöglichen?

6. Wie muss eine Soft- und Hardware-Architektur aufgebaut sein, um eine modellbasierte Testmethode in der Domäne verteilter BIS-Tests möglichst zu unterstützen?

7. Die generierten Testfälle und Testskripte müssen ausführbar sein. Die Testfälle müssen mit Testdaten versehen sein. Die zu entwickelnden Testskripte müssen sich in schon vorhandene Testskriptbibliotheken integrieren lassen.

8. Die zu entwickelnde Testmethode muss erweiterbar sein, um auf Veränderungen innerhalb der Systeminfrastruktur reagieren zu können.

9. Als Grundlage für die Wirtschaftlichkeitsanalyse der MBT-Testmethode werden die im Rahmen dieser Dissertation erlangten Erfahrungswerte und Kostenschätzungen genutzt.

1.5 Anforderungen an die zu entwickelnde Methode – Evaluierungskriterien

Die Anforderungen an die hier zu entwickelnde Methode bestehen in erster Linie darin, den Testabdeckungsgrad durch die Modellierung aller möglichen Pfade eines Geschäftsprozesses zu erhöhen. Eine weitere Anforderung ist die Reduzierung des manuellen Testaufwands und die gleichzeitige Erhöhung des Automatisierungsgrads von Testfällen. Zusätzlich soll dadurch eine Kostenersparnis erzielt werden, unabhängig vom notwendigen zu investierenden Initialaufwand. Hierzu werden Testsysteme aus Praxisumgebungen herangezogen.

Die Evaluierungskriterien leiten sich direkt aus den Anforderungen an diese Methode ab. Ein Evaluierungskriterium ist der Testabdeckungsgrad, welcher am Fallbeispiel (IT-gestützter Zahlungsabwicklungsprozess am Beispiel Debitorenprozess) untersucht wird. Gleichzeitig lässt sich ein weiteres Evaluierungskriterium aus der Testzeitersparnis ableiten. Dadurch lässt sich dann eine Aussage über die eingesparten Kosten pro Testlauf ermitteln. Der Umfang der Evaluierung und der Probe ist durch das Fallbeispiel dieser Dissertation gegeben. Ein abschließendes Kriterium ist die Nutzbarkeit dieser Methode, welche für jeden Benutzer aus IT- und Fachbereich mit einem Testwerkzeug aufruf- und ausführbar sein muss.

2 Stand der Technik

Dieses Kapitel soll einen Einblick in das Testen und in den aktuellen Stand der MBT-Test-Technologie geben. Dabei soll der erste Teil dieses Kapitels eine Einführung in das Thema des Softwaretestens bieten. Basierend auf diesem Wissen werden im zweiten Teil die Testansätze der Domäne der Embedded Systems und der verteilten BIS erläutert. Dabei werden sowohl die aktuellen theoretischen Ansätze als auch die bisher bekannten praktischen Ansätze (Tools) diskutiert. Abschließend bietet das Kapitel sowohl eine Zusammenfassung einer vom Autor dieser Dissertation durchgeführten Umfrage zum Thema MBT als auch einen Überblick über die aktuellen forschungs- und themenbezogenen Arbeiten. Daraus leitet sich die präzisierte Aufgabenstellung dieser Dissertation ab.

2.1 Was bedeutet testen?

Softwaretests werden durchgeführt, um Fehler aufzuspüren und um das Vertrauen in die korrekte Funktionalität eines Softwaresystems zu erhöhen. Dies ist einer der wichtigsten Bestandteile im Softwareentwicklungsprozess und macht häufig mehr als die Hälfte des gesamten Entwicklungsprozesses aus [7], [12]. Der Begriff des Testens lässt sich auf vielfältige Art und Weise definieren. Hier einige Definitionen führender Wissenschaftler auf dem Gebiet des Softwaretestens:

1. Spillner und Pol, Koomen [74] sehen bspw. im Testen eine Maßnahme des Qualitätsmanagements in der Softwareentwicklung. Je später Fehler entdeckt werden, desto aufwendiger ist ihre Behebung, woraus sich der Umkehrschluss ableitet: Qualität muss (im ganzen Projektverlauf) implementiert und deshalb nicht durch Tests „reingebracht“ werden.
2. Testen „bedeutet, eine gegebene Software zu einem gegebenen Zeitpunkt zu nehmen und überzeugend zu zeigen, dass sie den gegebenen Anforderungen entspricht“ [13].
3. Testen „ist die Ausführung eines Programms bzw. eines Programmsystems mit dem Ziel, Fehler zu finden“ [5].
4. Testen „ist die Ausführung eines Programms mit dem Ziel, seine Funktion zu bestätigen. Testen ist die Ausführung eines Programms mit dem Ziel, seine Zuverlässigkeit zu messen“ [4].
5. Testen „ist der Prozess, ein Programm mit der Absicht auszuführen, Fehler zu finden“. Aus dieser Definition ist zu folgern, dass ein Test dann erfolgreich war, wenn er einen Fehler gefunden hat [5].

Auch zeigt die Praxis, dass in der Softwareentwicklung eine gewisse Fehleranzahl immer als normal betrachtet wird und indirekt auch akzeptiert wird, solange die Software funktioniert. Im Gegensatz zur Industrie sieht [74] einen Unterschied: Hier werden Fehler nur noch in Extremsituationen erwartet.

Die grundlegendsten Testarten sind die Tests der inneren Struktur eines Programms sowie funktionale und nicht-funktionale Tests. Strukturtests sind Tests, die auf Modulebene

durchgeführt werden und ein konkretes Programmierwissen erfordern. Funktionale und nicht-funktionale Tests werden sowohl auf der Entwickler- als auch auf der Anwenderebene ausgeführt. Mit nicht-funktionalen Tests werden die Zuverlässigkeit und die Benutzbarkeit einer Applikation begutachtet, bspw. das Antwortzeitverhalten von Onlinebestellkatalogen. Hier werden in Testfällen spezielle Zeitobergrenzen festgelegt, die nicht überschritten werden dürfen. Simuliert werden diese Szenarien zumeist mit Testwerkzeugen.[5] Sowohl der Modultest als auch der nicht-funktionale Test reichen zum Testen einer Applikation nicht aus, da beide Tests keine Aussage über die Funktionalität der Software erlauben [10], [35]. Fehlende Funktionalitäten würden nicht erkannt werden. Hingegen zielen funktionale Tests auf die Validierung einer funktionalen Anforderung. Die funktionale Anforderung bietet Informationen, die es ermöglichen, Testfälle zu spezifizieren. Bei verteilten BIS-Softwaresystemen ist es unmöglich, die Abwesenheit von Fehlern zu beweisen, was auch nicht das Ziel des Testens ist [7], nach [74] wird sogar eine Anzahl an Fehlern in Softwaresystemen toleriert (solange diese nicht direkt auffallen). Deshalb werden in der Praxis so lange Testfälle manuell erstellt und ausgeführt, bis alle in einer Testplanung definierten Testendekriterien erfüllt sind, wobei zu jeder Funktionseinheit in einer funktionalen Spezifikation ein Testfall als ausreichend gilt [18]. Bei Modultests müssen geforderte Überdeckungsgrade erreicht werden [7]. Dabei werden Pfad-, Knoten- und Zweigüberdeckungen unterschieden. In einigen Branchen gelten diese Überdeckungskriterien als Kennzahl und als Basis für Systemabnahmen. Jeder Test unterliegt einem Testprozess und verschiedenen Teststufen.

2.1.1 Was für Testmethoden existieren?

Verfahren zur systematischen Generierung von Testfällen für ein Testobjekt werden als Testmethoden bezeichnet. Softwaretests sind oft eine analytische Maßnahme, die erst nach der Erstellung der Software durchgeführt wird. Die vorher geschriebenen Testfälle oder Testskripte werden somit auf die zu testende Software angewandt. Aber Qualitätsmaßnahmen können sich sowohl auf das zu testende Produkt selbst als auch auf den Prozess der Softwareentwicklung beziehen. Liggesmeyer [10] klassifiziert die Testmethoden in „dynamischer Test“ und „statischer Test“, beide gehören zu den analytischen Maßnahmen, da sie beide Artefakte analysieren, ob nun mit oder ohne Ausführung des Programms. Zusätzlich zu den Testmethoden existieren verschiedene Vorgehensweisen und Klassifikationen, welche Liggesmeyer in [10] beschreibt.

Konstruktive Maßnahmen (Statischer Test)

Statische Tests prüfen anhand der Spezifikation und der umgesetzten Implementierung die Korrektheit des Systems, ohne dieses auszuführen. Die konstruktiven Qualitätsmaßnahmen sorgen dafür, dass das entstehende Softwareprodukt bzw. der Softwareentwicklungsprozess vor der Entwicklung bestimmte Eigenschaften besitzt [10], [64]. Unterstützt werden diese Maßnahmen bspw. durch IT-Governance-Richtlinien [68], auf deren Basis Checklisten und Standards angefertigt werden können. Dazu gehören der Review sowohl der funktionalen

[5] Marktführer von Testwerkzeugen wie HP stellen für solche Tests Last- und Performance-Testwerkzeuge zur Verfügung.

als auch der technischen Spezifikationen. Auch gehört eine statische Codeanalyse zu den konstruktiven Maßnahmen. All diese Maßnahmen dienen dazu, schon vor Beginn der Entwicklung die funktionalen und die nicht funktionalen Anforderungen auf Qualitätskriterien zu überprüfen. Fehler sind in der Softwareentwicklung dadurch nicht vermeidbar, aber wie Balzert in [64] beschreibt: „Vorbeugen ist besser als Heilen“ und Fehler, die nicht gemacht werden, brauchen auch nicht behoben zu werden. Alle diese Maßnahmen dienen der Fehlervermeidung.

Analytische Maßnahmen (Dynamischer Test)

Dynamische Tests hingegen führen das zu testende Softwaresystem mit festgelegten Eingabedaten (Parametern) aus und überprüfen die erzielten Ausgaben (Ausgangsparameter) auf Konsistenz mit dem erwarteten Verhalten. Ein vollständiger Test, sprich der Test eines Systems auf Korrektheit hinsichtlich der möglichen Eingabewerte bspw. auf die verteilten BIS, ist nicht möglich. Es kann nur eine Aussage über die Korrektheit des Softwaresystems auf der Basis der dafür ausgewählten und durchgeführten Testfälle getroffen werden, wobei die Testfallgüte[6] entscheidend ist.

Analog zu den konstruktiven Maßnahmen können sich auch die analytischen Maßnahmen auf alle Softwareentwicklungsphasen oder direkt auf das Softwareprodukt [64], [10] beziehen. Bei den analytischen Verfahren werden analysierende und testende Verfahren unterschieden [64]. Im Rahmen dieser Dissertation fokussiert der Autor die testenden Verfahren. Analytische Maßnahmen werden allerdings erst dann vorgeschlagen, wenn vorher konstruktive Maßnahmen durchgeführt worden sind; auch kann beides parallel erfolgen. Wird die Software ausgeführt, um das Qualitätsniveau festzustellen, so handelt es sich hierbei um einen dynamischen Test. Der dynamische Test lässt sich nach Liggesmeyer weiter unterscheiden in Blackbox- und Whiteboxtests. Bei Whiteboxtests ist das „Innere“, bspw. der Quellcode, vorhanden und dient als Quelle für die Testfallerstellung. Verfahren, welche beim Whiteboxtest zum Einsatz kommen, sind bspw. Kontrollfluss-, Datenfluss- und architekturorientierte Testverfahren. Eine konkrete Erläuterung bietet [10].

Beim Blackboxtest wird das zu testende Softwaresystem rein funktional betrachtet. Es muss nach festgelegten Testkriterien auf eine Eingabe eine definierte und erwartete Ausgabe erfolgen. Im Rahmen dieser Dissertation fokussiert der Autor den Blackboxtest. Dieser funktionsorientierte Test beinhaltet als Quelle der Testfälle die funktionale Spezifikation eines zu testenden Softwaresystems. Um die Testfallermittlung methodisch zu unterstützen, bietet diese Klassifikation Methoden wie die Grenzwert- und die Äquivalenzklassenanalyse. Liggesmeyer beschreibt in [10] noch die Ursache-Wirkungs-Analyse und die anforderungsbasierte Testmethode als weitere Methoden, um Testfälle zu ermitteln. Die Dissertation setzt bei der Ermittlung von Testspezifikationen sowohl auf Äquivalenzklassen als auch auf Grenzwertanalysen, gepaart mit anforderungsorientierten Methoden, anhand derer Geschäftsprozesse nach ihrer Kritikalität bewertet werden. Unabhängig von den verschiedenen Teststufen, welche in der Testlehre existieren [7], [10], können diese Methoden eingesetzt werden.

---

[6] Bedeutet, wie die Testfälle inhaltlich beschrieben sind und ob diese auch so dokumentiert sind, dass sie Fehler finden können.

2.1.2 Der fundamentale Testprozess

Es gibt verschiedene Ansätze [5], [7], [10], [74], um den Software-Testprozess abzubilden, diese unterscheiden sich nur marginal. Grob lässt sich der Testprozess in einzelne Testphasen einteilen, wobei jede Testphase mit einer Planungsphase beginnen sollte.

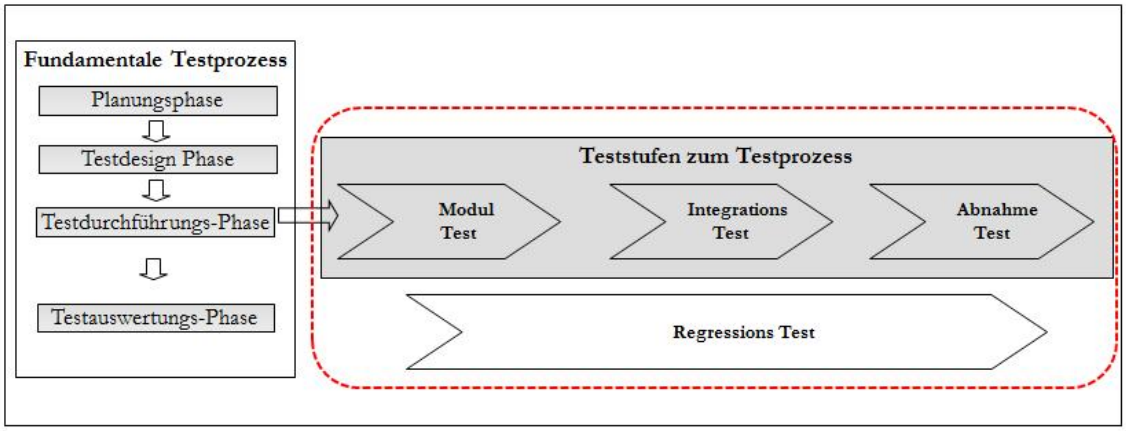


Abbildung 2: Fundamentaler Testprozess [7] (modifiziert durch den Autor dieser Dissertation – rot umrahmt)

In der Planungsphase wird eine Teststrategie festgelegt, welche die Besonderheiten des Softwaresystems untersuchen soll. Zusätzlich muss diese Strategie die für den Test erforderlichen Infrastrukturen berücksichtigen, um das Testziel bestmöglich zu erreichen. In der Testdesignphase müssen Testfälle aus funktionalen Spezifikationen formuliert und diese anschließend auf Konsistenz und Lücken untersucht werden. In der Testdurchführungsphase werden zu den erstellten Testfällen die passenden Testdaten hinzugefügt und in Sequenzen eingeteilt. Diese Sequenzen können einem Testscheduler[7] übergeben werden, der dann dafür verantwortlich ist, diese Testfälle in einer vordefinierten Reihenfolge auszuführen. Der Testscheduler spielt eine entscheidende Rolle, wenn mehrere Applikationen während des Testens miteinander agieren müssen.

In der Testdurchführungsphase kommen verschiedene Teststufen zum Einsatz. Der Modultest ist dabei die erste Teststufe im Softwareentwicklungsprozess und wird bestenfalls in einem Vier-Augen-Prinzip[8] von Entwicklern durchgeführt. Diese Teststufe unterscheidet sich vom Integrationstest dahingehend, dass Modultests i.d.R. auf Entwicklungssystemen erfolgen. Dabei fokussiert der Modultest die einzelnen entwickelten Softwarekomponenten ohne Integration in bestehende Systemlandschaften.

Der Integrationstest konzentriert sich dagegen auf die Integration der Softwarekomponenten in bestehende Systeminfrastrukturen und auf die daraus resultierenden applikationsübergreifenden Tests. Der Integrationstest ist nur aussagekräftig, wenn mit einer Simulation der Produktivumgebung getestet werden kann. Der Abnahmetest muss vom Kunden durchgeführt werden, hier sollte der Kunde prüfen, ob die angeforderten Funktionalitäten, wie er sie in der funktionalen Spezifikation spezifiziert hat, auch korrekt umgesetzt worden sind. Neben dem Integrations- und dem Abnahmetest ist der Regressionstest eine der zentralen „Teststufen“, auch wenn dieser Test in der Literatur und in der Praxis nicht als eine eigene Teststufe angesehen wird. Wenn man in der Softwaretechnik von ei-

[7] Ein Add-in in Testwerkzeugen, das die Ausführung von Testfällen automatisch steuert.
[8] Tests sollten nicht von dem Programmierer durchgeführt werden, der die Software auch entwickelt hat.

nem Regressionstest spricht, ist damit die Wiederholung von Testfällen gemeint. Ein Regressionstest stellt sicher, dass durch Änderungen in einer Softwareversion vorher existierende Funktionen nicht direkt oder indirekt beeinflusst werden, und zwar nach jeder Teststufe [7]. Da diese Tests bei jeder Änderung ausgeführt werden müssen, sollten diese Testfälle bestenfalls automatisiert werden [46].

Die Testauswertungsphase beschäftigt sich mit der Prüfung der in der Planungsphase definierten Testendekriterien. In dieser Phase werden Entscheidungen getroffen, bspw. ob Testendekriterien erreicht sind und wie mit Testfällen umgegangen wird, die aus Zeit- und Ressourcengründen nicht ausgeführt werden können. Zusätzlich werden in der Testauswertungsphase sämtliche Testergebnisse zusammengefasst und archiviert. Aufgrund des nahezu unendlichen Zustandsraums eines Softwaresystems und manchmal vielen unbekannten Verkopplungen mit vielen anderen Applikationen ist es unmöglich, alle Systemzustände abzubilden. Weil es nicht möglich ist, jedes Systemverhalten und das damit verbundene Sollverhalten zu prüfen, kann ein Test nur Fehler finden und niemals Fehlerfreiheit nachweisen. Ein Beispiel, das die Komplexität einer 100%igen Testabdeckung verdeutlichten soll, bietet Myers Glenford in [59].

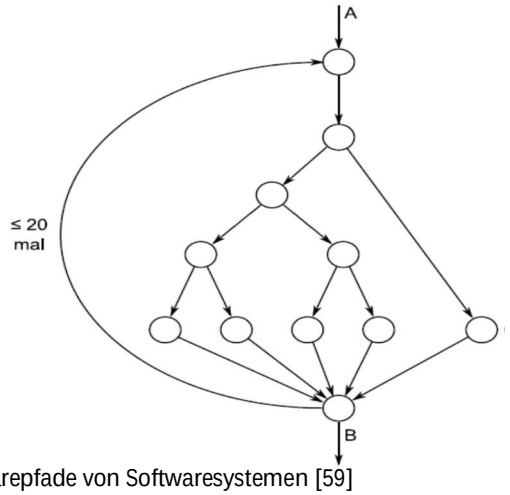


Abbildung 3: Softwarepfade von Softwaresystemen [59]

Das Beispiel zeigt ein Programm, das aus einer Schleife besteht, die 20 Mal durchlaufen werden kann, innerhalb der Schleife gibt es eine Reihe von Verzweigungen, die zu insgesamt fünf unterschiedlichen Pfaden durch die Schleife führen. Volker Knollmann [73] benutzt dieses Beispiel in seiner Dissertation und der Autor dieser Arbeit erweitert dieses Beispiel nun um folgende Berechnung, die die Anzahl der möglichen Kombinationen und Pfadüberdeckungen illustrieren soll: Bei einem einzelnen Durchlauf sind 5^1 Pfade möglich. Bei zwei Schleifendurchläufen sind es also 5^2 usw. Da jetzt in diesem Beispiel bis zu 20 Schleifendurchläufe möglich sind, müssen diese Kombinationen addiert werden. Damit lässt sich eine geometrische Reihe aufbauen: $5^1 + 5^2 + ... + 5^{20}$. Diese Berechnung führt zu der Aussage, dass 120 Billionen Pfaddurchläufe möglich sind. Das einfache Beispiel zeigt, dass es praktisch unmöglich ist, alle möglichen Ausführungspfade zu testen. Auch durch eine vollständige Pfadüberdeckung wird nur sichergestellt, dass jede Anweisung mindestens einmal ausgeführt wird, allerdings ohne ein Fehlverhalten[9] mit Hilfe von Äquivalenzklassen zu provozieren.

[9] Explizit falsche Testfälle formulieren, um zu überprüfen, ob das zu testende Softwaresystem darauf reagiert.

2.1.3 Whitebox- und Blackboxtests

Das Beispiel mit der Pfadüberdeckung, welche in der Praxis auch C2-Abdeckung [7], [10], [46] genannt wird, ist ein Kandidat für einen Whiteboxtest (Strukturtest). Die Whiteboxtest-Methode beinhaltet die Kenntnis der inneren Struktur des zu testenden Systems, diese kann jetzt bei der Auswahl der auszuführenden Testfälle berücksichtigt werden und beeinflusst dadurch entscheidend den Test. Neben der C2-Abdeckung existieren noch die etwas schwächeren Formen der Abdeckung (C1 und C0). Diese beiden Formen haben die vollständige Anweisungsüberdeckung bzw. Zweigüberdeckung zum Ziel.

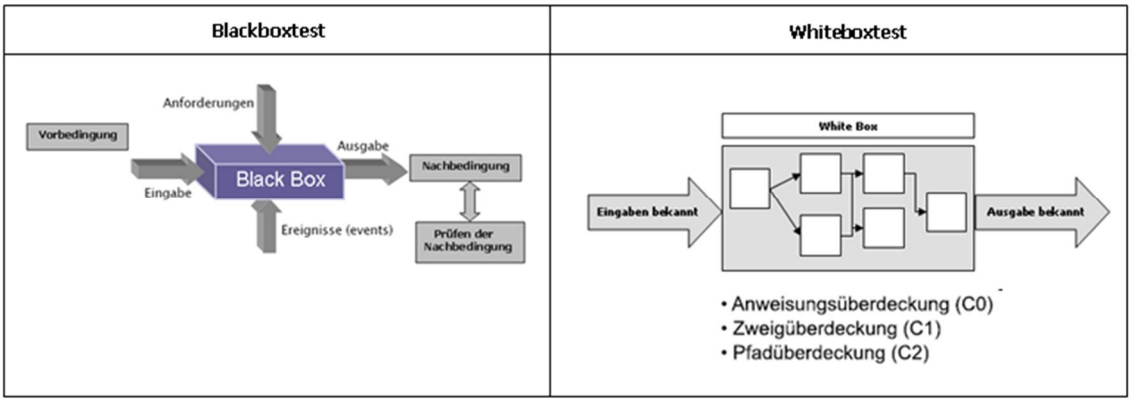


Abbildung 4: Schematische Darstellung des Unterschieds zwischen Blackbox- und Whiteboxtests

Beim Blackboxtest wird ohne die Berücksichtigung der internen Struktur des Systems getestet [10], dabei wird das zu testende System insbesondere auch nach seinem Schnittstellenverhalten beurteilt. Als Grundlage für die Testspezifikation dient die Anforderungsspezifikation. Der Whiteboxtest testet insbesondere die interne Struktur des Systems im Rahmen des Modultests.

Der Modultest liegt in der Hoheit der Programmierer, es muss sichergestellt werden, dass ein Vier-Augen-Prinzip[10] eingehalten wird. Der Programmierer sollte immer seinen eigenen Code durch einen anderen Programmierkollegen testen lassen. Dazu können Entwickler auf Testwerkzeuge zurückgreifen. Bestes Beispiel ist das Testframework JUNIT, hier wird für jede implementierte Klasse eine Testklasse erstellt. Alle Testklassen werden anschließend zu Testsuiten gebündelt und bieten somit gleichzeitig einen Datenpool an Regressionstestfällen. Bei jeder Änderung oder Erweiterung einer Applikation oder auch kleinerer Programmstücke müssen die Testfälle zur Überprüfung ausgeführt werden.

Auf der Basis der funktionalen Spezifikation kann ein Tester oder Kunde die entwickelten Komponenten auf die in der Anforderung spezifizierten Funktionalitäten testen. Zusammenfassend lässt sich sagen, dass sowohl Whitebox- als auch Blackboxtests zwei unterschiedliche Testziele verfolgen. Der Whiteboxtest konzentriert sich auf die entwickelten Quellcodes. Beim Blackboxtest kann keine Aussage darüber getroffen werden, welche Funktionen im Quellcode evtl. schon getestet worden sind, weil in dieser Testphase die Funktionalitäten der Applikation sowie die Schnittstellen, mit denen diese Applikation kommuniziert, im Vordergrund stehen. Beide Klassifikationen ergänzen sich bestenfalls, sollen sich aber niemals ersetzen. Eine Mischung aus beiden Verfahren bildet der Greybox-

[10] Wer ein Programmstück codiert, sollte nicht gleichzeitig Tester des Programmstücks sein.

test. Dieser spielt im Extreme Programming[11] eine entscheidende Rolle und trägt maßgeblich zum Erfolg dieser Methode bei. Beim Extreme Programming werden innovative Testansätze, wie der „Testfirst-Ansatz“, angewendet. Der Testfirst-Ansatz ist ein wichtiger Bestandteil des modellbasierten Testens, da auch beim MBT Testfälle schon in der Anforderungsphase modelliert werden müssen.

2.1.4 Testfirst-Ansatz am Beispiel JUNIT

In den meisten großen Firmen werden Softwaretests nicht durch Entwickler durchgeführt [4]. Der Grund hierfür ist die psychologische Hemmschwelle, eigene Fehler wahrzunehmen oder zu entdecken. So zeigt die Praxis, dass man die Testfälle unbewusst so formulieren wird, dass der eigene Quellcode korrekt ist [4]. Die Philosophie des Testfirst-Ansatzes ist es, Entwickler dazu zu verpflichten, bevor sie mit dem Programmieren einer Komponente beginnen, einen Testfall zu der zu entwickelnden Komponente zu schreiben. Das JUNIT soll dabei dem Programmierer das Erstellen von Testfällen erleichtern, indem es einen Rahmen bietet, solche Testfälle schnell zu formulieren und einzuprogrammieren. Ist der Testfall zu der Komponente erstellt, so kann mit dem Programmieren des Produktivcodes begonnen werden.

Abbildung 5: Testfirst-Ansatz

Es wird so lange programmiert, kompiliert und getestet, bis der Testfall erfolgreich ausgeführt worden ist. Da beim Testfirst-Ansatz die zu testende Funktion noch gar nicht implementiert ist, kann dem Programmierer auch kein Fehler nachgewiesen werden. Dies bietet jetzt die Möglichkeit, die Messlatte am eigenen Programmcode selbst hoch anzulegen und sich als guter Programmierer zu beweisen [4]. Werden im Testfirst-Ansatz keine Fehler während des Testens im Programmcode gefunden, so ist dies meistens darauf zurückzuführen, dass Entwickler sich mit der Thematik gründlich auseinandergesetzt haben, was alles schief gehen kann, und dadurch Fehler[12] frühzeitig vermieden werden [4]. Der Testfirst-Ansatz beschränkt sich nicht auf den Modultest, sondern kann auch auf den Integrations- bzw. Regressionstest, welcher auf Anwenderebene ausgeführt wird, übertragen werden. Wobei auch der Testfirst-Ansatz nicht garantieren kann, dass die zu entwickelnde Applikation fehlerfrei ist. Natürlich können auch Anforderungen und somit auch ein Testfall falsch formuliert sein. Dass dies nicht selten vorkommt, zeigen die 10 Risiken der Softwareentwicklung nach Boehm [123]. So ist die Gefahr, dass aus einer falsch formulierten Anforderung auch falsche Testfälle formuliert werden und dadurch die Applikation falsch getestet wird, sehr präsent und groß.

---

[11] Ein Vorgehensmodell, wie Software entwickelt werden kann – von zwei Pionieren der objektorientierten Programmierung entwickelt: Kent Beck und Erich Gamma.

[12] Dies garantiert aber auch keine Fehlerfreiheit.

Bei der Spezifizierung und Formulierung von Testfällen können Methoden wie Äquivalenzklassen und Grenzwertanalysen eingesetzt werden. Diese Methoden unterstützen eine methodische Testfallermittlung.

2.1.5 Testabdeckung durch Äquivalenzklassen und Grenzwertanalyse

Eine Äquivalenzklasse ist eine Teilmenge aller möglichen Eingaben in einem Testfall, so dass das erwartete funktionale Verhalten aller Elemente dieser Teilmenge identisch ist [7], [10]. Äquivalenzklassenbildung ist eine geeignete Methode, um aus Spezifikationen repräsentative Testfälle abzuleiten, und bildet die Basis für den Blackboxtest. In den meisten Fällen sind die Beispiele, die Äquivalenzklassen behandeln, die Äquivalenzklassen von Zahlen. Allerdings zeigt die Praxis, dass eine Beschränkung auf Zahlen allein nicht ausreichend ist. In fast allen Fällen muss auch mit Zeichen getestet werden, deshalb müssen dementsprechend Zeichen auch Bestandteil einer gültigen Äquivalenzklasse sein.

Die Grenzwertanalyse benutzt die Testdaten von Äquivalenzklassen, welche die Werte an den Rändern berücksichtigen. Bei Zahlen können via Grenzwertanalyse Repräsentanten bestimmt und gleichzeitig geprüft werden. Bei einem Text ist dies nicht so einfach möglich, da es je nach Klassenbildung keine „harten“ Grenzen mehr gibt. Was hier vonnöten wäre, ist eine vollständige Prüfung einer ungültigen Äquivalenzklasse [59], [68]. Solange man sich auf die Werte beschränkt, welche über die Tastatur einzugeben sind, ist dies noch überschaubar. Werden andere Eingabegeräte für das Testen benutzt, z. B. Kassensysteme mit Hilfe von Scannern, so können die Äquivalenzklassen schnell unüberschaubar werden. Bei der Grenzwertanalyse werden gezielt Grenzwerte in Kombination mit falschen Werten eingegeben, um die Robustheit und die Fehlertoleranz des Systems zu prüfen. Im Falle numerischer Werte werden die unteren und die oberen Grenzwerte erprobt. Im Falle von Texten werden falsche und leere Zeichenfolgen eingegeben. Außerdem sollen sowohl zulässige als auch unzulässige Tastenkombinationen und Mausklicks erprobt werden. Dazu müssen gültige und ungültige Äquivalenzklassen gebildet werden.

2.1.6 Schwachstellen heutiger Testfälle

Die Schwachstellen vieler klassischer Testfälle liegen darin, dass diese unvollständig spezifiziert sind [4], dennoch zeigen [53], [54] und [59], wie ein vollständiger Testfall mit allen Eingabemöglichkeiten spezifiziert werden kann. Zusätzlich müssen unterschiedliche Randbedingungen berücksichtigt werden, die Einfluss auf den Systemablauf haben. Ein etwas abgeändertes Beispiel aus [59] zeigt, welcher Zeitaufwand in Anspruch genommen werden muss, um alle unterschiedlichen Abläufe (Pfade) eines Programmes zu testen. Dort bestand eine Software aus 120 Billionen möglicher Pfaddurchläufe. Allein wenn für jeden Pfad eine geschätzte Dauer von ca. einer halben Minute angesetzt wird, übersteigt der Testaufwand jegliche Dimension. Dieser Aufwand kann bspw. ohne Testwerkzeugunterstützung kaum in einer vertretbaren Zeit bewältigt werden. Neben den Blackbox- und Whitebox-„Methoden“ zur systematischen Testfallermittlung darf beim Testen der gesunde Menschenverstand nicht vergessen werden. So weiß ein Tester aus seinen Erfahrungen um die

Schwachstellen bei der Programmierung und beschreibt Testfälle, die genau diese Situation prüfen sollen, bei denen eine Wahrscheinlichkeit auf Fehler besteht. Solche intuitiven Tests finden in der Praxis nicht immer Anwendung. Viel zu oft wird der Fokus nur auf die systematische und strukturierte Ermittlung von Testfällen gelegt, anstatt auch intuitive Tests einzubinden, welche auf den Erfahrungen von Testern basieren [143].

2.1.7 Testfälle ausreichend spezifizieren

Was oft unterschätzt wird, ist die Notwendigkeit, in diesem Rahmen Testfälle zu definieren, die auch auf Sicherheit abzielen sollen, bspw. ob eine SQL-Befehlssequenz (SELECT *; FROM *; WHERE) in ein Textfeld eingegeben werden darf und ob diese auch etwas bewirken (ausführen) kann. Dies erfordert ein strukturiertes Vorgehen und die genaue Spezifizierung, was ein Testfall alles können bzw. abdecken muss.

Unter einem Testfall versteht man ein spezielles Paket von Vorbedingungen (Anforderungen an den Zustand des Systems vor der Durchführung des Testfalls), Eingaben (Eingabewerte, zur Eingabe notwendige Aktionen) und erwarteten Ergebnissen (Ausgabewerte, neuer Zustand des Systems, (Fehler-) Meldungen etc.), das auf das Testobjekt angewandt wird. Dabei müssen in einem Testfall folgende Eigenschaften berücksichtigt werden:

- ✓ Gültige Eingaben in das System, welches dem Normalfall entspricht.
- ✓ Ungültige Eingaben (Fehlerfall), auf die das System reagieren muss, Eingaben, die gerade noch gültig sind, oder nicht mehr gültige Eingaben, wie 29.02. in einem Nichtschaltjahr.

So müssen gute Testfälle Vorbedingungen definieren, die erfüllt sein müssen, bevor Aktionen oder Funktionsabläufe stattfinden. Werden diese Vorbedingungen verletzt, kann ein Funktionsaufruf entweder nicht oder nur fehlerhaft ausgeführt werden. Auf verletzte Vorbedingungen muss aus Systemsicht entsprechend reagiert werden. Als Beispiel muss eine Buchung in einem System erfolgt sein, bevor diese storniert werden kann.

Auch müssen Testfälle Sonderfälle betrachten, wie den Grenzbereich bei Integritätsbedingungen von Vorbedingungen und Eingabewerten (z. B. Datumswechsel, Schaltjahre). Als ein weiterer Sonderfall ist der Fehlerfall zu simulieren, wobei bewusst Eingabewerte verwendet werden, die unter den gegebenen Vorbedingungen in einem Fehler terminieren. Dadurch wird die Robustheit des Systems gegen Eingabefehlern getestet. Beispielsweise bei der Stornierung eines nicht gebuchten Belegs.

Ein Testfall muss strukturiert und systematisch durchgeführt werden. Zusätzlich muss dieser mit intuitiven Eigenschaften erweitert werden, um viele Fehler mit angemessenem Aufwand zu finden und dabei gleichzeitig unnötige Tests und eine Inflation an Testfällen zu vermeiden.

Um einen nahezu perfekten Testfall zu spezifizieren, sollte zu jedem Test eine Checkliste entworfen werden, welche Fehlererwartungen enthält:

- ✓ Division durch Null ist ein klassisches Beispiel.
- ✓ Zahlenüberläufe erzeugen, bspw. bei einem Kassensystem, um festzustellen, welcher höchstmögliche Einkaufswert übergeben werden kann. Kann diese Zahl verarbeitet werden? Gibt es Zahlenüberläufe und Rundungsfehler?

- ✓ Wenn es sich um eine Web-Applikation handelt, sollte getestet werden, was passiert, wenn mitten bei der Verarbeitung einer Transaktion der „Refresh-Button“ gedrückt wird.

Diese Tests können mit Hilfe von Äquivalenzklassen und mit der Grenzwertanalyse spezifiziert werden [68].

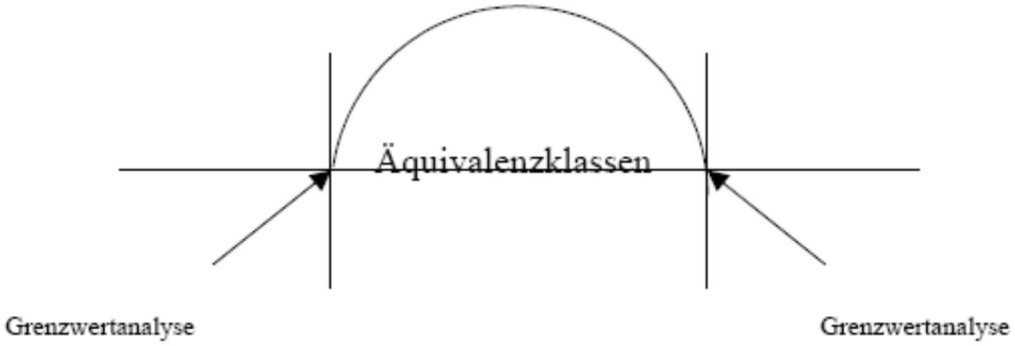


Abbildung 6: Äquivalenzklassenbildung als Grundlage für die Grenzwertanalyse

Ein weiteres Instrument, auf das im Rahmen dieser Dissertation nicht näher eingegangen werden kann, ist das Error Guessing. Dieses Instrument beruht auf der Erfahrung und dem Wissen des Testers. Prinzipiell legt man eine Liste möglicher Fehler oder fehleranfälliger Situationen an und definiert damit Testfälle; eine ausführliche Darstellung des Error Guessing bietet [59].

Für die Bildung von Äquivalenzklassen bieten sich heute verschiedene Tools an, wie die der Firma SQS[13] mit SQS-Testprofessional. Diese Tools vereinfachen automatisiert erheblich die Spezifikation von Testfällen mit automatisch generierten Testdaten.

Zusammenfassend lässt sich sagen, dass mit Hilfe der Grenzwertanalyse getestet werden kann, ob das System auf alle stellvertretenden Impulse richtig reagiert. Für das MBT bieten sowohl die Äquivalenzklassenbildung als auch die Grenzwertanalyse ein Instrument, das bestimmt, welche Pfade in einem Modell mit welchen Testdaten getestet werden können.

2.1.8 Softwaretests im Softwareentwicklungsprozess

Für die systematische Durchführung großer Softwareentwicklungsprojekte hat sich seit Jahren das V-Modell etabliert. Auch in der Domäne verteilter BIS hat das V-Modell weitgehend herkömmliche Entwicklungsmodelle, wie zum Beispiel das Spiralmodell oder das Wasserfallmodell, ergänzt oder gar ersetzt. Das ursprünglich im Jahr 1992 veröffentlichte V-Modell wurde später von dem V-Modell 97 und dann im Jahr 2005 von dem V-Modell XT mit einigen Erweiterungen abgelöst [132]. XT bedeutet „Extreme Tailoring“ und hilft dabei, das V-Modell bspw. auch auf kleine Projekte und Produkte zu skalieren.

[13] SQS steht für Software Quality Systems, dies gilt als Marktführer im Bereich Beratung im Testing von Anwendungen.

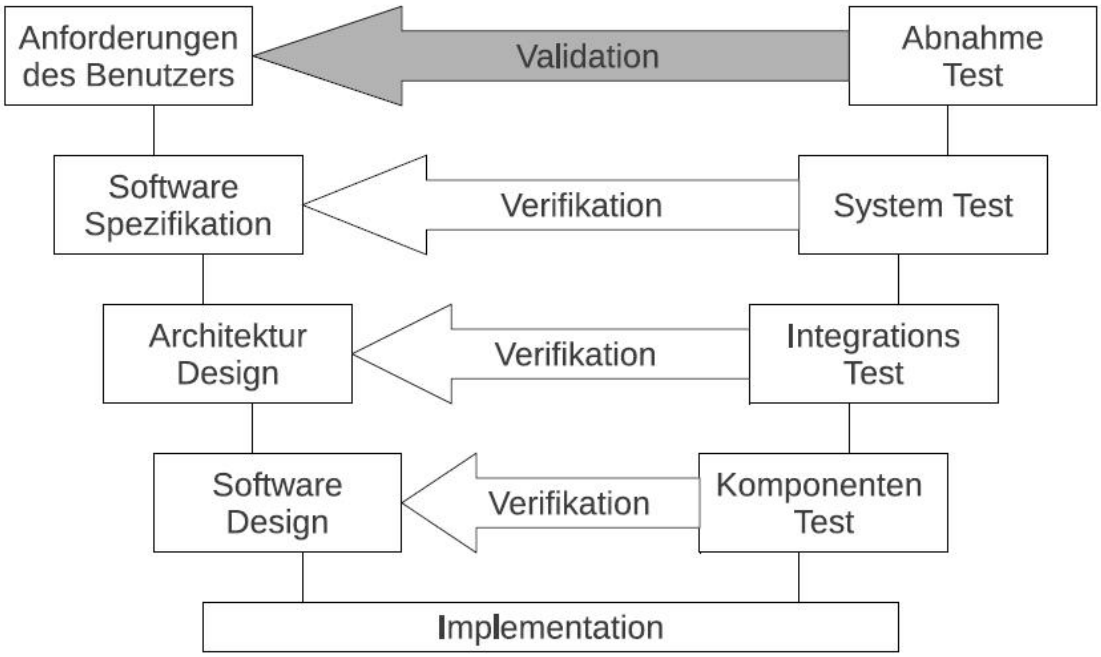


Abbildung 7: V-Modell nach Boehm, modifiziert durch den Autor der Dissertation

Die Abbildung zeigt eine für die Domäne der verteilten BIS typische Herangehensweise an die Entwicklung von Software gemäß dem V-Modell. Zuerst werden die Erwartungen an die Software mit Hilfe von Anforderungen in funktionalen Spezifikationen festgehalten. Ausgehend von den Anforderungen werden detaillierte Spezifikationen und Designkonzepte für das zu realisierende System erstellt. Handelt es sich um eine Erweiterung eines bestehenden Systems, so werden in verteilten BIS sogenannte Business-Blueprints erstellt. Auf der Basis dieser Konzepte werden Erweiterungen und Anpassungen an Systemen vorgenommen. Die Ergebnisse eines Business-Blueprints müssen dann in einer funktionalen Anforderung geeignet integriert werden.

Im Falle der modellbasierten Entwicklung kann ein ausführbarer Programmcode direkt aus den Spezifikationen erzeugt werden. Allerdings wird auch heutzutage noch fast vollständig insbesondere der Programmcode in der Domäne verteilter BIS manuell implementiert. Sind die ersten Module fertiggestellt, beginnt auf dem rechten Ast im V-Modell die Testphase.

Dabei werden zuerst Modultests durchgeführt und die Module werden basierend auf den Ergebnissen der Testläufe untersucht und ggf. überarbeitet, bis die vordefinierten Testendekriterien erreicht sind. Die einzelnen Komponenten werden dann zu größeren Einheiten zusammengeführt, um die Kommunikation und die Interaktion zwischen den Komponenten zu überprüfen (Integrationstests). Neben den klassischen Softwareentwicklungsverfahren existieren heute agile Softwareentwicklungsverfahren wie das Extreme Programming und Scrum[14]. Der Unterschied in den Vorgehensweisen liegt nicht nur im Prozess, sondern auch in der Hervorhebung der Testphasen. Beim Extreme Programming bildet der Test-first-Ansatz nicht nur die Phase des Testens, sondern die dort definierten Testfälle werden zusammen mit Entwicklern und Fachbereichen geschrieben und dienen gleichzeitig auch als Anforderungsspezifikation. Mehr zum Thema Software-Vorgehensmodelle bieten [124]

[14] Scrum ist die etwas schwächere Form der agilen Softwareentwicklungsprozesse (bspw. gegenüber Extreme Programming).

und [125]. Im Zusammenhang mit Software-Vorgehensmodellen und Softwaretests werden die Begriffe Validierung und Verifikation unterschieden.

2.1.9 Verifikation und Validierung

Allgemein versteht man unter Softwaretests die Verifikation und Validierung einer Softwarekomponente oder eines Softwaresystems [7], [8]. Beides sind im IT-Qualitätsmanagement Aspekte von Prüfverfahren. Ziel dieses Vorgangs ist es, die Qualität bzw. die Korrektheit einer Software zu überprüfen. Thaller [69] sieht in der Verifikation die Überprüfung einer Software am Ende einer Entwicklungsphase hinsichtlich der an die Software gestellten Anforderungen, bezüglich eines Standards oder bezüglich der Ergebnisse der vorhergehenden Phase. D. h., die Verifikation hinterfragt, ob das System mit der funktionalen Spezifikation übereinstimmt. Bei der Validierung handelt es sich um die Prüfung der Frage, ob die an das Softwareprodukt gestellten Anforderungen umgesetzt sind. Um die beiden Begriffe besser zu unterscheiden, folgt ein Beispiel anhand eines Drei-Gänge-Menüs. Das Menü wird nachweislich ganz genau nach Vorgabe gekocht = Verifizierung. Die meisten Gäste sind voll und ganz mit dem Ergebnis zufrieden = Validierung: Kundenanforderung erfüllt. Einige Gäste hätten lieber eine Suppe statt eines Salates als Vorspeise = Validierung: Kundenanforderung nicht erfüllt. Es wird die Auswahl „Salat oder Suppe“ angeboten = Prozess wird neu verifiziert.

Alle Kunden sind zufrieden = Prozess ist validiert, Kundenanforderung erfüllt.

Die Verifizierung weist nach, dass die erzeugten Komponenten der Spezifikation aus der Anforderungsentwicklung genügen. Das bedeutet, dass die Tests von unten nach oben (Bottom-up) durchgeführt werden. Demgegenüber liefert die Validierung eine Bestätigung dafür, dass das Produkt den Merkmalen genügt, die der Endanwender in Form einer funktionalen Spezifikation gefordert hat. Dies impliziert einen von oben nach unten laufenden Systemtest.

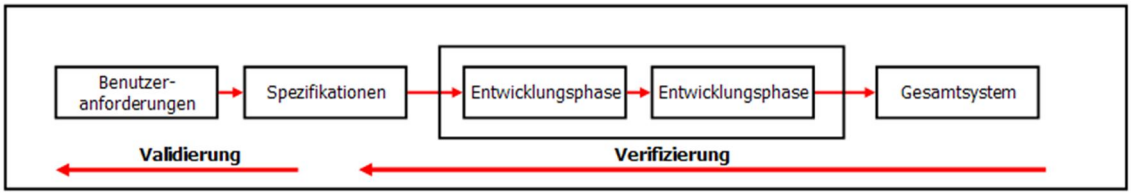


Abbildung 8: Validierung und Verifizierung

Zusammenfassend gesagt, beinhaltet die Validierung die Überprüfung, ob das System das richtige „tut“, wobei ein Experte die Spezifikation oder das gesamt System prüft [69]. Die Verifizierung prüft, ob das System das richtige „tut“, sprich es prüft gegen die Spezifikation. Um eine hohe Qualität der Software in verteilten BIS zu garantieren, ist die Durchführung von umfangreichen Integrations- und Regressionstests unabdingbar. Aufgrund der steigenden Systemkomplexität ist eine Testautomatisierung entscheidend für die Realisierbarkeit der Validierung mit vertretbarem Aufwand.

2.2 Automatisierung von Softwaretests

Die Testautomatisierung ist die Durchführung von ansonsten manuellen Testtätigkeiten durch Testautomaten [133]. Testautomaten sind Programme, die wiederum andere Programme testen, es handelt sich um sogenannte Metaprogramme. Die Bandbreite für die Testautomatisierung erstreckt sich über die Tätigkeiten zur Überprüfung der Software im Entwicklungsprozess bis hin zu den unterschiedlichen Entwicklungsphasen und Teststufen sowie über die entsprechenden Aktivitäten im Integrationstest und im Kundenabnahmetest. In der Praxis findet eine Automatisierung der Testdurchführung in mehreren Teststufen [10], [77], [133] bereits eine breite Anwendung.

Ein Nachteil der klassischen Testautomatisierung (Capture & Replay) liegt darin, dass diese „nur“ die manuellen Tätigkeiten eines Testers übernimmt, nicht aber dessen Intuition. Diese ist jedoch maßgeblich für die Qualität des Softwaretests selbst [133], daher sollte, wenn auf Testautomatisierung zurückgegriffen wird, dieser Umstand nie ignoriert werden. Auch soll die Testautomatisierung nicht dazu beitragen, dass Tester wegrationalisiert werden, sondern sie soll ein Instrument bieten, das es dem professionellen Softwaretester erlaubt, eine große Anzahl an Testfällen mit vernünftigem Aufwand und in angemessener Zeit zu bewältigen.

Ein Beispiel ist die automatisierte Ausführung von Modultests in der Programmiersprache Java. Mit Hilfe der Testumgebung JUNIT [47] werden dabei Whiteboxtests programmiert und automatisiert ausgeführt.

2.2.1 Automatisierung der Modultests mit JUNIT (Whitebox) in Java

Das JUNIT ist ein Testframework, um Java-Programme zu testen. Analog dazu gibt es in C++ das CUNIT. Das JUNIT bietet komfortable Möglichkeiten, um Tests zu schreiben und automatisiert auszuführen. JUNIT lässt sich einfach in gängige Entwicklungsumgebungen wie ECLIPSE integrieren. Mittlerweile gibt es die Version 4.8 und höher, wobei mit Version 3.8 JUNIT der Durchbruch gelang [4]. Der Nachteil ist, dass Anwender eine gute „intime“ Kenntnis des Frameworks besitzen müssen, um damit arbeiten zu können. So kommen in dem JUNIT-Framework in Version 3.8 verschiedene Design Patterns zum Einsatz, die das Erstellen von Testfällen erheblich vereinfachen [4]. Was aber gleichzeitig bedeutet, dass man sich mit dem Thema Patterns auseinandersetzen muss.

Patterns bieten ein einheitliches Sprachvokabular, damit können sich Entwickler untereinander besser austauschen. Sie beinhalten das Wissen und den Erfahrungsschatz von Entwicklern, so dass Einsteiger bei einem Programmierproblem nicht immer bei null beginnen müssen, sondern auf bewährtes Wissen und Programmierlösungen von Experten zurückgreifen können. Mehr zu Design Patterns bietet [4].

Testfälle identifiziert das JUNIT-Framework mit Hilfe von Namenskonventionen und der *Reflection* API von Java [41], [70]. Mit Hilfe der Instrumente der *Reflection*, bspw. der *Introspektion*, kann nicht nur während der Laufzeit auf den Bauplan des Programms zugegriffen werden, sondern es können auch identifizierte Methoden zur Ausführung gebracht werden. Mit der Version 4 des JUNIT Frameworks kamen grundlegende Erneuerungen sowie An-

notationen hinzu. So erfolgt die Identifizierung der Testfälle nicht mehr über Namenskonventionen, sondern es werden Testmethoden im Programm mit Annotationen versehen. So erkennt das JUNIT Framework, welche Methoden Testfälle sind und welche nicht [41], [148].

Mit der JUNIT-Version 4.4 kam anschließend eine besondere Verbesserung, die die Testdichte erhöhen soll [126], und zwar können ab JUNIT-Version 4.4 Testfälle zusätzlich parametrisiert und mittels Testdaten direkt im Coding versorgt werden. Das Framework sorgt dafür, dass der Testfall mit allen möglichen Kombinationen ausgeführt wird.

2.2.2 Automatisierung der Modultests in SAP mittels ABAP-Unit (Whitebox)

Ein etwas anderes Beispiel bieten Modultests in der Programmiersprache ABAP. ABAP ist eine Programmiersprache für eines der mächtigsten betrieblichen Informationssysteme, die es in der Softwaretechnologie gibt, SAP[15] [144]. Der Einsatz von automatisierten Quellcode- und Modultests bildet erst die Grundlage moderner Softwareentwicklung. Deshalb ist es möglich, in SAP Netweaver, welches auch die Entwicklungsumgebung der ABAP-Modultests bildet, Testfälle zu programmieren und automatisiert auszuführen. Im Unterschied zu JUNIT und Java können in SAP die Modultests in zwei verschiedenen Klassentypen ausgeprägt werden, in globalen und lokalen Testklassen. Da ABAP auch eine objektorientierte Programmiersprache ist, kann eine globale Testklasse (Schemaklasse analog zur abstrakten Superklasse in Java) gebildet werden. Eine Schemaklasse bildet den Rumpf für andere Klassen, die von ihr erben. Die lokale Klasse ist dann eine Klasse, die bspw. von einer Schemaklasse abgeleitet sein kann, dadurch herrscht automatisch eine enge Kopplung zwischen der globalen und der lokalen Klasse. Das Framework, um Modultests für ABAP-Klassen zu entwickeln, ähnelt sehr dem JUNIT-Framework von Java [127], [128]. Dabei besteht ein Testfall generell aus drei Phasen:

- Setup (): Methode, mit der das Initialsetup für einen Test erstellt werden soll. An dieser Stelle werden z. B. die für den Test benötigten Objekte erzeugt, die nötigen Verbindungen zu Servern, Datenbanken usw. hergestellt sowie benötigte Mockobjekte[16] generiert.
- Test (): Methode, die die tatsächliche Testausführungslogik inklusive aller Parameter und Kombinationsmöglichkeiten beinhaltet. Der Aufruf dieser Methode ist im Gegensatz zu Java hart verdrahtet. Die Test()-Methode beinhaltet eine weitere Methode Testresult (), die dafür verantwortlich ist, alle Ergebnisse der Tests zu sammeln und auszuwerten.
- TearDown (): Diese Methode spielt im ABAP im Gegensatz zum JUNIT-Framework eine wichtige Rolle. Diese Methode ist dafür verantwortlich, alle Objektgeflechte, die in der Setup()-Methode aufgebaut wurden, wieder abzubauen und somit die geblockten Ressourcen wieder freizugeben. In Java erledigt das der Garbage Collector automatisch.

[15] Systeme Anwendungen Programme.
[16] Mock-Objekte werden in der testgetriebenen Softwareentwicklung „Dummy“-Objekte oder Attrappen genannt, die als Platzhalter für echte Objekte innerhalb von Unit-Tests verwendet werden.

Zusammenfassend ist festzuhalten, dass jeder Modultestfall in ABAP aus den drei Methoden (Setup(), Test(), TearDown()) besteht. ABAP bietet zusätzlich den Vorteil, Testklassen automatisch generieren zu lassen, so dass diese Methoden als Gerüst immer schon vorhanden sind (siehe auch Patterns [4]).

Das ABAP-Unit-Test-Framework ist Bestandteil der Entwicklungsumgebung und ist genauso wie das JUNIT frei verfügbar und kann leicht in jede Entwicklungsumgebung integriert werden [127]. Der Einsatz solcher in ABAP entwickelten Modultests ist vielfältig, diese können für jede entwickelte Komponente zum Einsatz kommen. Der Testfirst-Ansatz kann analog zu JUNIT auch hier Verwendung finden. Eine ausführliche Einführung in das Thema ABAP-Unit-Tests bietet [127].

Die beiden Modultestwerkzeuge JUNIT und ABAP-Unit-Test konzentrieren sich auf Whiteboxtests. Das Anwenden und Ausführen dieser Testwerkzeuge erfordert ein gewisses Maß an Programmiererfahrungen sowie einen sicheren Umgang mit Entwicklungsumgebungen. Dabei beschränkt sich die Testautomatisierung nicht nur auf die Modultestebene, sondern erstreckt sich auch auf die Anwenderebene.

Die Durchführung von Blackboxtests übernehmen Personen, die wissen, was das System leisten soll, aber nicht, wie diese Leistung erbracht wird. Dabei fokussiert der Test ausschließlich die Anwenderebene und Benutzerinteraktionen. Auch heute schon existiert in der Praxis eine Reihe von Testwerkzeuglösungen, die die fachlichen Anwendertests unterstützen. Zwei davon sind HP Quicktest Professional und IBM SQA Robot. Einen ausführlichen Überblick über die am Markt befindlichen kommerziellen und nicht-kommerziellen Testwerkzeuge bietet [129]. Eine etwas erweiterte und detaillierte Übersicht bietet Danny Faught in [71], [72].

2.2.3 Automatisierung von Funktionstests (Blackboxtests)

Testwerkzeuge auf Funktionstestebene, wobei der Funktionstest in der Praxis auch als Synonym für Integrations- und Regressionstests steht, bieten bestimmte Funktionalitäten (Programme und Bibliotheken), um den Testprozess für den Endanwender zu vereinfachen [34]. Es gibt wenige Testwerkzeuge, die gleichstark alle Aufgaben des automatisierten Testens übernehmen [71]. Vielmehr kristallisieren sich immer mehr Testwerkzeuge heraus, die sich auf eine Aufgabe wie das GUI-Testen spezialisieren. Solche Testwerkzeuge fokussieren das Testen der grafischen Benutzeroberflächen. Die Besonderheiten beim Testen von Benutzeroberflächen sind, dass die Benutzeroberflächen der Anwendung ihre Funktionalität reflektiert und dass die Anwendung über die GUI bedient wird. Somit wird sowohl die Präsentationsschicht als auch die Anwendungslogik getestet. Die Besonderheiten, die bei Funktionstests zu beachten sind, fasst [42] zusammen und ergänzt werden diese Besonderheiten durch den Autor dieser Dissertation.

- Immer wiederkehrende Regressionstests: Die Entwicklungen von GUIs unterliegen meistens einem agilen Softwareentwicklungsprozess [42] und befinden sich daher im ständigen Wandel. Dadurch wird hier ein ständiger automatisierter Re-

gressionstest benötigt, der unter Umständen[17] auch immer mit neuen Testdaten durchgeführt werden muss.

- Soll-Resultatbestimmung: Bei einem klassischen Test werden die erwarteten Ergebnisse für einen Testfall [34] beschrieben und mit den durch den Test erhaltenen Ergebnissen verglichen. So müssen bei einem Funktionstest streng genommen alle Testschritte, d. h. jede getätigte Benutzerinteraktion, geprüft und validiert werden.
- Jeder unerwartete Zustand macht i.d.R. eine weitere Testausführung unmöglich, weil die Benutzerinteraktionen aufeinander aufbauen. Im Gegensatz zu Unit-Tests, denn hier wird für jeden Testfall ein eigenes Objektgeflecht mittels der Setup-Methode erzeugt. Auf Basis dieses Geflechts kann der Testfall seine Tests ausführen, nach dem Test kann eine weitere Klasse ein neues Objektgeflecht erzeugen und diese können wiederum auf ein frisches Objektgeflecht ihre Testfälle ausführen. D. h., die Testfälle können immer unabhängig voneinander ausgeführt werden. Bei Blackboxtests ist so etwas nur mit einem sehr hohen Aufwand möglich.

Grafische Benutzeroberflächen bieten immer mehrere Möglichkeiten, um eine Aufgabe zu lösen. Problematisch ist es dann, wenn es mehrere Pfade gibt, um einen Testfall auszuführen. Alle diese Pfade müssen bei der Programmierung der automatisierten Testskripte berücksichtigt werden.

Im Gegensatz dazu gibt es in JUNIT ab der Version 4.4 die sogenannten „Theories“, die das Parametrisieren von Testfällen erlauben, und somit können bspw. diese Tests mit allen möglichen Kombinationen ausgeführt werden, aber auch hier ist der Entwickler in der Pflicht, auch alle möglichen Kombinationen bei der Programmierung zu berücksichtigen. Um alle möglichen Kombinationen sowohl beim Modultest als auch beim Funktionstest zu ermitteln, kann auf Äquivalenzklassenbildung und auf die Grenzwertanalysen zurückgegriffen werden. Die Benutzerfreundlichkeit kann nicht getestet werden, sondern diese kann gesondert über einen empirischen Weg ermittelt werden [42].

Der Einsatz eines Testwerkzeugs auf der Anwenderebene ist sinnvoll, wenn dieses Werkzeug dazu beiträgt, den Aufwand von Routinetestaktivitäten zu senken und gleichzeitig den Testabdeckungsgrad zu erhöhen. Dieses Vorgehen impliziert wiederum die Erhöhung der Wahrscheinlichkeit, Fehler zu entdecken. Abschließend sollen nicht die Nachteile der klassischen Testautomatisierung von Funktionstests verschwiegen werden.

- Der Initialaufwand, um einen Testfall zu automatisieren, ist im Gegensatz zum manuellen Test um ein Vielfaches höher [29], [42].
- Capture & Replay: Testskripttechniken, die viele Hersteller von Testwerkzeugen versprechen, funktionieren in der Praxis nicht immer erfolgreich [67]. Testskripte müssen immer zusätzlich mit sehr hohem Aufwand erweitert werden, während Frameworks gebaut werden müssen, um die entwickelten Testskripte/Programme wiederverwendbar für andere Testfälle in Bibliotheken zu verlagern.
- Auf Anwenderebene werden Tester mit sehr hohem fachlichem Wissen über die zu testende Applikation benötigt. Testautomatisierer, die sowohl fachlich die Geschäftsprozesse beherrschen als auch in der Lage sind, Testfälle zu automatisieren, sind meistens Exoten [29].

[17] Wenn die Testdaten nicht einfach zurückgesetzt werden können.

- Wartungsproblem: Testskripte müssen ständig einem Wartungszyklus unterliegen, denn sonst kann es passieren, dass evtl. Anpassungen an die zu testende Software nicht in die Testskripte einfließen und somit die Testskripte nicht mehr funktionieren oder, im schlimmsten Fall, einfach falsch testen [4].

Die genannten Nachteile verdeutlichen, warum die klassische Testautomation[18] nicht einfach umzusetzen ist. Die Empfehlung des Autors ist, dass Testautomationsprojekte generell als Softwareprojekte aufgesetzt werden müssen. So soll sichergestellt werden, dass entwickelte Testskripte automatisch einem Softwarelebenszyklus und gleichzeitig einem Wartungsprozess unterstellt sind. Werden Änderungen am zu testenden Objekt angestrebt, so muss das Testskript dementsprechend mit angepasst und versioniert werden. Das Wartungsproblem ist auch heute noch einer der häufigsten Gründe, warum Testautomationsprojekte in verteilten BIS scheitern [67], [77]. Wobei auch die Kosten für die Wartung und für den Initialaufwand beim Erstellen von Testskripten nicht unterschätzt werden dürfen [77]. Eine in der Literatur häufig zitierte Studie von Dustin [144] zeigt, dass der Testvorbereitungsaufwand im automatisierten Fall etwa doppelt so hoch ist wie im manuellen Fall. Jedoch ist die Testdurchführung im automatisierten Testfall viel geringer. Da im Regressionstest in der Regel mehrere Testdurchläufe geplant sind, eignen sich besonders Regressionstests für eine Testautomation.

Die richtige Auswahl der zu automatisierenden Testfälle ist auch ein Erfolgsfaktor für eine Testautomation, so sollten nur Testfälle von Geschäftsprozessen automatisiert werden, die kurzfristig nicht verändert werden müssen. Regressionstestfälle zeichnen sich dadurch aus, dass sie mindestens schon einmal zum wiederholten Mal ausgeführt worden sind. Die Kategorisierung im Regressionstestfall sollte dabei immer in enger Abstimmung zwischen Entwicklung, Testern und dem zuständigen Fachbereich erfolgen. Ein Trend der letzten Jahre ist, dass sich führende Testwerkzeughersteller wie HP, IBM, aber auch Experten der objektorientierten Programmierung wie Harrold in [135] Gedanken machen, wie Regressionstestfälle automatisiert identifiziert werden können.

2.2.4 Automatisierung von Regressionstests – Notwendigkeit einer modellbasierten Testmethode beim Blackboxtest

Hierzu entwickelten Harrold und die Mitautoren aus [135] das Verfahren „RETEST“. Dieses Verfahren ist nicht nur eines der ersten für Java, sondern auch eines der präzisesten für objektorientierte Software, nicht nur aufgrund der Tatsache, dass alle Konstrukte moderner objektorientierter Sprachen unterstützt werden. Hier werden nach bestimmten mathematischen Berechnungen und Kriterien Regressionstestfälle auf Modultestebene berechnet. Für Funktionstests sind solche Berechnungen momentan noch kein Forschungsgegenstand [135], obwohl Forschungsbedarf herrscht [75]. Daher bleibt an dieser Stelle nur die manuelle Identifizierung von Regressionstestfällen. Das Prinzip des Regressionstests verdeutlicht folgende Abbildung:

[18] Die klassische Testautomatisierung beruht auf der Programmierung der Testskripte mittels Capture&Replay.

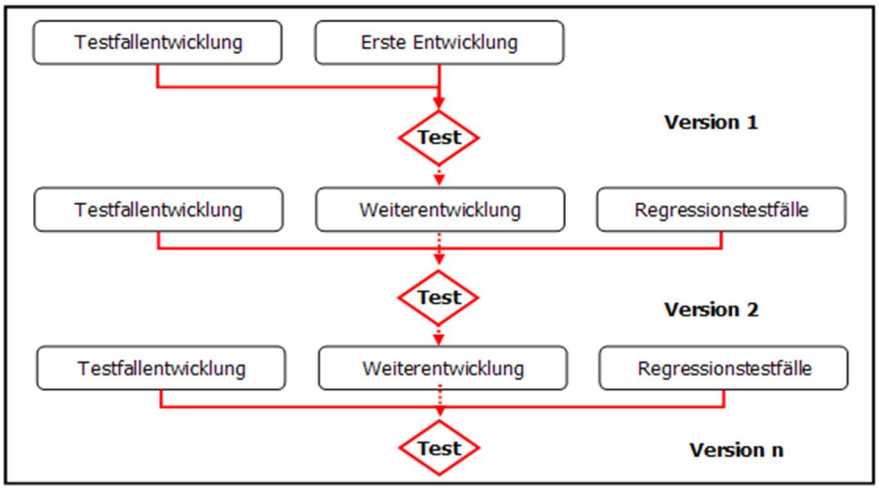


Abbildung 9: Prinzip Regressionstest

Der Regressionstest ist nach der IEEE der Standard für die Software-Engineering-Terminologie. Er dient dem erneuten Test einer bereits getesteten Software nach deren Modifikation mit dem Ziel, nachzuweisen, dass durch die vorgenommene Änderung/Erweiterung keine neuen Fehler eingebaut wurden [134]. Eine Änderung kann bspw. eine einzelne Klasse, eine Komponente, ein Teilsystem oder ein ganzes Anwendungssystem sein. Je größer das zu testende System, desto größer der Umfang des Regressionstests. Werden Systemschnittstellen verändert, so muss das ganze System mit allen Testfällen erneut getestet werden. Eine Reduzierung des Regressionstests auf die kritischsten[19] Testfälle wird als „risikobasierter Test“ bezeichnet und ist seit vielen Jahren Gegenstand zahlreicher Forschungsarbeiten [135]. Mit risikobasierten Tests werden keine 100%igen Testabdeckungen erreicht, sondern hier werden in einer Analyse zunächst einmal alle Testfälle nach Wichtigkeit priorisiert [133].

In der Praxis fehlt meistens die Zeit, um alle spezifizierten Tests, die notwendig sind, auszuführen. Deshalb muss eine sinnvolle Auswahl an Testfällen getroffen werden, um so viele kritische Fehlerwirkungen zu finden, wie möglich. Das Ziel einer Priorisierung in Testphasen ist: Beim Abbruch des Tests zu einem beliebigen Zeitpunkt soll das bis dahin beste Ergebnis erreicht werden. Die Kriterien für Priorisierungen sind nicht gleichgültig, sie hängen meist vom Projekt, vom Anwendungsbereich und von Kundenwünschen ab [145]. Bei unkritischen Softwarekomponenten sollte keine Zeit verschwendet werden. Insbesondere bei Softwaresystemen mit vielen Schnittstellen ist die Auswahl zweckmäßiger Testfälle ebenso notwendig wie deren Priorisierung für die Reihenfolge der Testausführung. Ein Ansatz zur Priorisierung von Testfällen ist das risikobasierte Testen. Ein risikobasierter Testplan soll dabei sicherstellen, dass Teile einer Software umso intensiver getestet werden, je höher das Risiko ist, dass diese Teile beim Einsatz der Software zu einem nicht vernachlässigbaren Schaden führen [68], [69]. Beim risikobasierten Testen werden Systemfunktionen je nach Risikostufe mit unterschiedlichen Testverfahren und Testtiefen abgedeckt. Wobei Komponenten und Bereiche mit höherem Risiko bei der Testplanung eine höhere

[19] Kritische Testfälle testen für das Unternehmen kritische Geschäftsprozesse. Jeder Geschäftsprozess, welcher einem Unternehmen bei einem Ausfall oder einer Fehlfunktion finanziellen Schaden zufügt, kann als kritischer Prozess bezeichnet werden. Dieser liegt aber auch dann vor, wenn durch Fehlfunktionen ein Kundenverlust entstehen kann.

Priorität erhalten und entsprechend frühzeitig und intensiv getestet werden. Daraus resultieren die Kriterien, die Regressionstestfälle erfüllen müssen:

- Sie sollten die kritischen Pfade eines Softwaresystems testen.
- Der kritische Pfad deckt gleichzeitig einen kritischen Geschäftsprozess ab und ist dadurch höher priorisiert als unkritische Geschäftsprozesse.
- Die Summe aller kritischen Geschäftsprozesse macht eine Prozessüberdeckung möglich. Der Begriff der Prozessüberdeckung im Zusammenhang mit einer Testabdeckung befindet sich nach der Recherche des Autors dieser Dissertation in keiner Literatur und ist durch den Autor dieser Dissertation definiert als die Summe aller kritischen Prozesse eines Softwaresystems.

Bestenfalls sind alle kritischen Bereiche der Software im Regressionstest abgedeckt, was aber auch keine Fehlerfreiheit in den Testfällen selbst garantiert, daher müssen Regressionstestfälle immer auf Konsistenz und Fehler untersucht werden [63], bspw. durch gezieltes Einbauen von Fehlern in Testfälle oder in Softwaresysteme.

Ein weiterer Grund, der Regressionstests nötig macht, sind Refaktorisierungsmaßnahmen. Diese werden zunehmend auch in großen Softwaresystemen durchgeführt. Sie kommen dann zum Einsatz, wenn Software für große Erweiterungen vorbereitet werden muss. Refaktorisierung zeichnet sich dadurch aus, dass Programme umstrukturiert werden können, ohne das Verhalten dieser Software zu verändern. Um zu überprüfen, ob das Verhalten der Software durch die Refaktorisierung auch tatsächlich nicht beeinflusst worden ist, werden automatisierte Regressionstests herangezogen, die vor der Refaktorisierung schon vorhanden sein müssen. Deshalb sind Begriffe wie Regressionstest, Testfirst-Ansatz und Refaktorisierung eng miteinander gekoppelt [4]. Friedrich Steimann ist Professor für Programmiersysteme an der Fernuniversität Hagen und sein Forschungsschwerpunkt liegt neben den Programmiersystemen auch in der Refaktorisierung großer Softwaresysteme. In diversen Publikationen[20] untersucht sein Institut, inwiefern sich gängige Refaktorisierungen aus dem Katalog von Martin Fowler [130] umsetzen lassen.

Sowohl das Change-Management[21] als auch Refaktorisierungsmaßnahmen verlangen immer wieder nach umfangreichen Tests, deshalb ist es sinnvoll, diese Regressionstestfälle zu automatisieren [75]. In verteilten BIS werden klassische Testautomationsansätze wie das „Capture & Replay“ eingesetzt, um verschiedene kritische Testprozessszenarien zu automatisieren. Vor allem Geschäftsprozesse verteilter Applikationen, die via Schnittstellen miteinander agieren, müssen in immer kürzer werdenden Produktzyklen getestet werden.[22] Bei verteilten BIS wie ERP-Systemen kommt noch erschwerend hinzu, dass für die einzelnen Applikationen verschiedene Produkt- und Releasezyklen existieren, die einen zeitglei-

[20] Internet: www.fernuni-hagen.de, unter: Fakultät für Mathematik und Informatik, Lehrgebiet Programmiersysteme, Friedrich Steimann, Publikationen.

[21] Änderungen an Software, bedingt durch Anforderungen des Tagesgeschäftes.

[22] Das sind Erfahrungswerte aus der Praxis, wo ständige Releasewechsel, Updates und gesetzliche Änderungen, die in die verschiedensten Systeme eingepflegt werden müssen, dazu führen, dass ständig Testphasen durchlaufen werden müssen, wodurch gleichzeitig die Testzeiträume immer kürzer werden. Vergleiche dazu auch ein Interview mit Marion Schäfer, Marketing-Managerin bei ABAS Software, AGQEM Supplier 2012 S. 82–83. Generell lässt sich dieser Trend aber auch in allen anderen Branchen verfolgen. Die Herausforderungen, denen sich heutige Unternehmen stellen müssen, zwingen Unternehmen dazu, ständig ihre Software zu verbessern und anzupassen, um wettbewerbsfähig zu bleiben. Und das Testen bleibt das Instrument, um sicherzustellen, dass die Software so funktioniert, wie die Spezifikation es vorgesehen hat.

chen Integrationstest unmöglich machen. Trotzdem müssen diese Applikationen zunächst einmal unabhängig von den angebundenen Systemen getestet werden, bevor die applikationsübergreifenden Tests erfolgen können. Die einzelnen, unabhängig voneinander geführten Tests generieren einen großen Testaufwand, der ohne eine funktionierende Testautomation kaum zu bewältigen wäre.

Sowohl fachliche als auch technische Release-Upgrades implizieren Änderungen an einem Softwaresystem, welche wiederum Änderungen an vorhandenen Testfällen erfordern. Testfälle, die evtl. schon automatisiert sind, müssen folglich auch angepasst werden. Die Praxis zeigt, dass der Anpassungsaufwand nicht unerheblich ist. In [77] illustriert der Autor dieser Dissertation ein Praxisbeispiel einer Testautomation der Metro Group. Der Beitrag verdeutlicht die Grenzen klassischer Testautomationsansätze (Capture & Replay). Im schlimmsten Falle muss ein Testskript neu programmiert werden [67], [77]. Auch die Anzahl möglicher Testszenarien ist bei verteilten BIS üblicherweise sehr umfangreich und muss aus der Perspektive der Qualitätssicherung eine hohe Testfallgüte enthalten, welche die Anzahl verbleibender Fehler auf ein akzeptables Minimum reduziert [8]. Zusätzlich erfordern die wachsende Komplexität und die Funktionalität verteilter BIS neue Ideen und Methoden, um den immer kürzer werdenden Produktzyklen entgegenzutreten. Das Ziel ist es, die Testabdeckung in den verteilten BIS zu erhöhen und gleichzeitig alle für einen erfolgreichen Test notwendigen Testfälle auch in einem kurzen Testzeitraum auszuführen. Beim Thema Testabdeckung ist es wichtig, zu erwähnen, dass der Autor dieser Dissertation neben den in der Literatur [1] gängigen Überdeckungskriterien wie Pfad-, Knoten- und Kantenüberdeckungen ein weiteres Überdeckungskriterium definiert: die Prozessüberdeckung. Die Prozessüberdeckung basiert auf Erfahrungswerten von Fachbereichen aus dem täglichen Praxisbetrieb dieser Softwaresysteme. Als Basis der Prozessüberdeckung wird die Pfadüberdeckung herangezogen, dabei werden nur die Pfade durchlaufen, welche eine höhere Priorität aufweisen. Für eine Prozessüberdeckung ist es nicht erforderlich, alle möglichen Pfade einer Software zu durchlaufen, sondern nur die geschäftskritischen Prozesse [9]. Mit Hilfe der MBT-Methode können kritische Pfade im Testmodell identifiziert und priorisiert werden (siehe dazu Kapitel 2.3).

So gelten in der Domäne der Embedded Systems MBT-Testansätze als gelebter Standard [25], [30]. Hier wird gezielt die MBT-Testmethode auf der Basis von UML-Diagrammen im Modultest eingesetzt. Die Herausforderung besteht jetzt darin, einen vergleichbaren Ansatz auf die Domäne der verteilten BIS zu übertragen. Wichtig hierbei ist, dass nicht Modultestfälle in den Vordergrund treten, sondern Tests auf der Anwenderebene (funktionale Tests): Kundenabnahmetests. Der entscheidende Unterschied liegt darin, dass hier keine Entwickler oder Techniker die Tests ausführen und verantworten, sondern Fachbereiche ohne ausgeprägten IT-technischen Hintergrund.

Erste theoretische Ansätze bei ERP-Systemen arbeiteten [12] und [13] aus. Hier wurden zwei konkrete Beispiele vorgestellt, wie abstrakt ein Testmodell angefertigt und getestet werden kann. Wichtig in diesem Zusammenhang ist, dass das MBT in der Domäne der verteilten BIS nicht das Testen neu erfinden soll, sondern vorhandene Technologien und Methoden nutzen soll, um basierend auf dem klassischen und fundamentalen Testprozess [7], [8] eine erweiterte, innovativere Testmethode zu unterstützen und zu integrieren. Dies soll es ermöglichen, eine schnelle, effiziente und qualitativ hochwertige Software auszulie-

fern. Auch soll das Testen dadurch mehr in den Vordergrund rücken und nicht nur in der Theorie und in Lehrbüchern zentraler Bestandteil des Softwareentwicklungsprozesses sein, sondern vor allem ein gelebter Standard.

2.3 Modellbasiertes Testen (MBT)

Modellbasierte Test- und Entwicklungsverfahren sind heute Gegenstand vieler aktueller Forschungs- und Entwicklungsarbeiten [25]. Sie alle haben gemeinsam, dass ein zentraler Bestandteil der Entwicklung Modelle sind, aus welchen Codes oder auch Testfälle generiert werden können. Das MBT stellt dabei schon seit Jahren ein neues Paradigma in der Entwicklung von Embedded-Systemen dar [37]. Dabei fokussiert das MBT die Automatisierung von Aktivitäten in einer sehr frühen Entwicklungsphase. Dazu gehört die Automatisierung von Testfällen, welche aus Testmodellen gewonnen werden. Das ist in einigen Softwaredomänen schon eine fortgeschrittene Methode, mit der das Testen früh im Entwicklungsprozess integriert wird.

In anderen Domänen, wie bspw. bei den verteilten BIS, werden nach wie vor aus umgangssprachlich formulierten funktionalen Anforderungsspezifikationen Testfälle manuell erstellt. Das MBT konzentriert sich auf die Erstellung eines Testmodells auf der Basis einer funktionalen Spezifikation, anhand derer Testfälle automatisch aus einem von einem Testdesigner (in der Praxis gibt es dafür die Rolle des Business-Analysten) entworfenen Testmodell generiert werden sollen [22]. Damit ist der erste Teil der MBT-Philosophie erfüllt, der zweite Teil sieht vor, aus den jetzt fertig generierten Testfällen Testskripte zu generieren. Die Testskripte müssen präpariert werden, damit eine automatisierte Ausführung wiederum mit Testwerkzeugunterstützung möglich wird. Präparieren bedeutet an dieser Stelle, nicht nur zu überprüfen, ob die Testdaten, die ein Testskript beim Testlauf benötigt, auch vorhanden sind. Sondern es muss untersucht werden, ob die generierten Testskripte aus manuellen Testfällen lauffähig sind. Geschieht sowohl die Generierung von Testfällen als auch die Ausführung automatisiert, so kann sie als MBT-Testmethode bezeichnet werden [22].

2.3.1 Modellbasiertes Testen in der Domäne der Embedded Systems

Die bis zu diesem Zeitpunkt klassischen Methoden zur Entwicklung und zum Testen eingebetteter Softwaresysteme wurden den spezifischen Herausforderungen bspw. der Automobilbranche nicht mehr gerecht. Das FIRST-Forschungsteam am Fraunhofer-Institut in Berlin beschreibt [18], wie sehr das modellbasierte Testen sowie das automatisierte Generieren von Testfällen aus Modellen in der Entwicklung von Steuergeräten in Kraftfahrzeugen mittlerweile Einzug gehalten haben. Dabei existieren im Bereich von Embedded-Systemen schon einige ausgereifte Technologien (Werkzeuge), die das modellbasierte Testen unterstützen, beispielsweise bei der automatischen Generierung von Testfällen aus Modellen oder bei der Testauswertung.

Automobil-Industrie

Bevor eine Zeile Code geschrieben wird, sollte ein konzeptionelles Modell des Systems vorhanden sein [36]. In manchen Industriebereichen ist die Dokumentation der erstellten Software durch solche Modelle sogar die Voraussetzung für eine Abnahme durch den Kunden, z. B. in der Automobilindustrie durch Automotive SPICE®. Automotive SPICE ist eine domänenspezifische Variante des internationalen Standards ISO/IEC 15504 (SPICE). Der Zweck von Automotive SPICE ist die Bewertung der Leistungsfähigkeit der Entwicklungsprozesse von Steuergerätelieferanten in der Automobilindustrie [113].

Die zunehmende Verbreitung der softwaregesteuerten Systeme im Automobil [49] und deren steigende Komplexität forderten neue Methoden bei der Entwicklung und beim Test der Software [60]. Seit Anfang 2000 ist hier ein Paradigmenwechsel im Vorgehen der Entwicklung erkennbar. Die konventionelle Entwicklungsweise wird zunehmend durch einen modellbasierten Prozess ersetzt. Dr. Sadegh Sadeghipour beschreibt in [38], wie in der Automobilbranche automatisierte Strukturtests in Kombination mit modellbasierter Softwareentwicklung umgesetzt werden können. Dabei wird Simulink/Stateflow als Modellierungsumgebung eingesetzt. Für die automatische Codegenerierung wird *TargetLink*[23] und als Testumgebung *MTest* benutzt [35]. Die automatische Testauswertung erfolgt über *MEval*. TargetLink, *MTest* und *MEval* sind Produkte der dSPACE GmbH. *MTest* erlaubt die Messung verschiedener erreichter Überdeckungskriterien der Testdurchläufe [114]. Ein Pilotprojekt stellte die Daimler AG zusammen mit Prof. Peleska [20] vor. In diesem Beitrag wurden die Problemstellungen und Lösungen für automatisierte, modellbasierte Hardware-in-the-Loop-(HIL)-Tests von Fahrzeug-Steuergeräten beschrieben [50], [51]. Skizziert wird, welche Lösungen die Wissenschaft heute für Probleme der automatischen Testfallerstellung und für die zugehörige Testdatengenerierung für eingebettete nebenläufige Echtzeitsysteme zu bieten hat: Tatsächlich können neuere Verfahren, die auf Methoden des Constraint Solving [20] beruhen und dabei Techniken aus der mathematischen Numerik und der künstlichen Intelligenz einsetzen, Testfallgenerierungsprobleme für Systemmodelle mit einer Komplexität lösen, die den Anforderungen der Industriepraxis gerecht wird [52].

Einen weiteren Ansatz bietet die Berner und Mattner Systemtechnik GmbH; diese spezialisierte sich in den letzten Jahren zunehmend auf das modellbasierte Testen von Automobilsteuergeräten. Dadurch entstand das Werkzeug Modena, was einen Ansatz zum modellbasierten Testen implementiert. Dabei unterstützt dieses Tool die Wiederverwendung von Applikationsmodellen und die leichte Veränderbarkeit bzw. Anpassbarkeit der Testmodelle. Dieses Werkzeug befindet sich bei mehreren Automobilzuliefererfirmen im Einsatz und bietet eine bewährte Lösung.

Medizintechnik

Auch in der Medizintechnik spielen Software und das modellbasierte Testen eine entscheidende Rolle. Software steuert die unterschiedlichsten Geräte, wobei deren Qualität vor ihrem Einsatz auch intensiv und gewissenhaft geprüft werden muss [45]. Es liegt auf der Hand, dass bei der Entwicklung von Softwarekomponenten, die eine Herz-Lungen-Maschine oder einen Herzschrittmacher steuern, die Gewährleistung von Zuverlässigkeit allerhöchste Priorität [38], [39] hat. Bis es ein medizinisches Gerät auf den Markt schafft, muss es, ähnlich wie sicherheitsrelevante Steuergeräte in der Automobilbranche, die Prüf-

[23] http://www.dspace.com/de/gmb/home/products/sw/pcgs/targetli.cfm.

verfahren mehrerer Qualitätsstufen durchlaufen [113]. Darüber hinaus werden noch klinische Tests, z. B. an Tieren und Menschen, durchgeführt, bevor das Gerät Marktreife erreicht. Dieses wird dann durch den TÜV oder durch die US-amerikanische Food & Drug Administration (FDA) zertifiziert und zugelassen [82], [86]. Dafür ist ein detaillierter Nachweis aller Qualitätssicherungsmaßnahmen erforderlich, weswegen die Entwicklung von medizinischen Geräten teuer und langwierig ist. Im Gegenzug haben die entwickelten Geräte einen sehr hohen Qualitätsstandard. Für die Hersteller solcher Geräte ist es wichtig, die Produkte lange einsetzbar, leicht erweiterbar und wartbar zu halten. Darüber hinaus muss die Bedienbarkeit der Geräte nicht nur benutzerfreundlich sein, sondern auch falsche Handhabungen durch das Personal soweit wie möglich ausschließen, um im Notfall lebenswichtige Funktionen zu übernehmen [18]. Wie ein Praxisbeispiel aus der Medizintechnik aufgebaut ist, zeigt ein Projekt des Fraunhofer-Instituts für Rechnerarchitektur und Softwaretechnik, abgekürzt: FIRST [39], [40]. Dabei wird illustriert, wie Steuergeräte, die ohne Frage lebensnotwendige Funktionen erfüllen, qualitativ mit Hilfe von MBT-Methoden getestet werden. Modellbasierte Verfahren mit ihrem hohen Grad an Automatisierung bewirken einen erheblichen Effizienzgewinn in der Entwicklung und in der Qualitätssicherung, selbst wenn sie in einem späteren Produktlebenszyklus existierender Systeme zum Einsatz kommen.

Bahntechnik

Aber nicht nur in der Automobilbranche und in der Medizintechnik, sondern auch in der Bahntechnik führen die modellbasierten Verfahren zu einer hohen Produktivität bei der Softwareerstellung, woraus eine hohe Qualität bei den entwickelten Produkten resultiert. Dazu veröffentlichte Volker Knollmann [73] im Rahmen seiner Dissertation ein Verfahren, wie sich aus einer Komponente eines Softwaresystems ein kombiniertes Testfall- und Systemmodell abbilden lässt. Die Motivation seiner Arbeit war die Einführung des europaweiten Zugsicherungssystems ECTS (European Train Control System). ECTS zeichnet sich dadurch aus, dass es die Sicherheitsverantwortung für Menschen, Umwelt und Material übernimmt. Es bestand ein Interesse daran, neue Ansätze für die Planung, Umsetzung und vor allem für den Test sicherheitsrelevanter Applikationen zu entwickeln, da die alten Methoden sich nur noch mit sehr viel Aufwand realisieren ließen. Der Nutzen dieser Arbeit sollte es sein, Anforderungsbeziehungen automatisiert auszuwerten, um aus Modellen einen Testabdeckungsgrad zu bestimmen. Dabei griff Knollmann die Forschungsthemen aus dem Umfeld der Modellierungssprache UML auf und kombinierte sie zu einem durchgängigen Verfahren für die konsistente System- und Testfallbeschreibung in einem einzigen UML-Modell. Wobei nur die grundsätzliche Eignung und Realisierbarkeit einer Methode vorgestellt wurde, d. h., diese Methode wurde nur entwickelt und nicht erprobt, deshalb ist sie noch ausbaufähig [73], [53].

Wie in der Einleitung aufgeführt, führt der Wettbewerbsdruck Softwarehersteller dazu, in neue Innovationen zu investieren, um so einen Vorsprung zu erzielen und sich Wettbewerbsvorteile zu sichern [61]. Die eingebettete Software in elektronischen Systemen wird im Zuge des technischen Fortschritts immer komplexer und gleichzeitig steigen die Anforderungen an Qualität und Zuverlässigkeit [62]. Der Test ist und bleibt in absehbarer Zukunft das wichtigste Mittel zur Sicherung der Qualität und der Zuverlässigkeit der Software. Um diese Tests modellbasiert auszuführen, bedarf es ausgereifter Testwerkzeuge.

2.3.2 Modellierungstools und Testfallgeneratoren – Produktüberblick

Für die Generierung von Testfällen aus Testmodellen existiert am Markt eine Reihe von Testwerkzeugen, basierend auf der Unified Modeling Language (UML). Zumeist werden diese in Modultests eingesetzt [30]. Generell muss bei MBT-Testwerkzeugen zwischen reinen Modellierungswerkzeugen, Testfallgeneratoren und Reportgeneratoren unterschieden werden. Hinzu kommen klassische Testdatengeneratoren, die auch für die Äquivalenzklassenbildung herangezogen werden können, welche hier außer Betracht bleibt.

Testfallgenerator	Unterstützte Modelltypen	Hersteller
Qtronic	UML-Zustandsdiagramme und QML	Conformiq
MBTSuite (getmore)	UML-Aktivitätendiagramme	Seppmed
SmartTesting	UML-Diagramme	TestDesigner
TestBench	UML-Zustandsdiagramme (bedingt Aktivitätendiagramme)	Imbus AG
Telelogic Rhapsody	UML-Diagramme	Telelogic
Modena	UML-Diagramme	Berner & Mattner
eXept	UML-Diagramme	eXept Software AG
All4Tec	Markov-Ketten (bedingt auch UML-Zustandsdiagramme)	All4Tech

Die Auswahl gibt einen Überblick über die für den Markt bedeutendsten kommerziellen MBT-Testwerkzeuge und deren Hersteller, angelehnt an [30], und erweitert diese Aufstellung um das Produkt MBTSuite der Seppmed GmbH, welches die Basis des Lösungsansatzes dieser Dissertation bildet. Nach Recherchen und Interviews[24] mit Herstellern und Anbietern von MBT-Testwerkzeugen sind dies tatsächlich die Anbieter, welche sich zurzeit den MBT-Markt teilen [30]. Daraus resultiert die Motivation, gerade diese Hersteller kurz vorzustellen.

Conformiq Qtronic

Die Firma Verify Technology stellte 2010 die Conformiq Suite vor, mit der Testfälle modelliert werden können, präziser durch den *Conformiq Modeler*. Mit einer weiteren Engine können die modellierten Testfälle in ausführbare Testfälle überführt werden. Der Vorteil der Conformiq Tool Suite ist, dass die Modellierung und die Generierung von Testfällen aus einer Hand erfolgen. Die Conformiq Tool Suite besteht aus dem Conformiq Modeler (zum Erstellen der Modelle) und dem Conformiq Designer (für die automatische Testgenerierung). Conformiq Designer ist ein Eclipse®-basiertes Tool für die Automatisierung der Testfallerstellung für funktionale Tests (Blackboxtests). Das Tool generiert auf der Basis von Modellen automatisch Testfälle, Testpläne und ausführbare Testskripte in Industrie-Standard-Formaten wie Python, TCL, TTCN-3, C, C++, Visual Basic, Java, JUNIT, Perl, Excel, HTML, Word, Shell Scripts. In Deutschland wird Qtronic durch Verifysoft Technologie vertrieben. Conformiq sieht den SmartTest von TestDesigner als größte Konkurrenz

[24] Interviews mit Mitarbeitern bei Seppmed und SQS. Obwohl SQS kein eigenes Werkzeug für das modellbasierte Testen anbietet, tendieren die Empfehlungen für das Testen von Unternehmensgeschäftsprozessen zur MBTSuite, vor dem Hintergrund, dass sie leichter an BPMN anzukoppeln ist als andere Testwerkzeuge.

zu seinem Produkt Qtronic [30]. Er bietet ein mächtiges Werkzeug zur Generierung von Testfällen und zu deren automatisierter Ausführung.

Allerdings stellt sich der Modellierungsprozess als insgesamt sehr „entwicklungslastig“ dar, insbesondere die textuelle Modellierung mit der UML entspricht prinzipiell einem Entwicklungsprojekt. Dementsprechend müssen Anwender etwas länger und mit technischem Fokus geschult werden. Möglich ist es, Modelle im UML-Zustandsdiagramm und in QML (Qtronic Modeling Language) zu erstellen. Qtronic konzentriert sich deutlich auf den Bereich der Testfallgenerierung und auf die anschließende Testautomatisierung [30], was gleichzeitig den Schwerpunkt dieses Produktes ausmacht. Das Produkt ist nicht nur ein Testfallgenerator, sondern auch eine Ausführungseinheit, sprich, es kann auch direkt die generierten Testfälle in Testskripte überführen, dort bilden sie einen Coderumpf, der dann geeignet ausprogrammiert werden muss. Auch eine Anbindung an strategisch wichtige Testmanagementtools, wie z. B. an das HPQC, ist gegeben.

Eine weitere Komponente ist zum Conformiq Modeler hinzugekommen, RUDI[25]. Dieser soll neben QML und den dort verfügbaren Zustandsdiagrammen das Modellieren von Testprozessen in Aktivitätsdiagrammen, ähnlich der BPMN, ermöglichen. Ziel ist es, Testprozesse so nutzerfreundlich und integrativ wie möglich zu modellieren. Die Daimler AG setzt bspw. bei der Entwicklung von Softwaremodulen für automatische Fahrzeuggetriebe die modellbasierte Lösung der Conformiq Suite zum Testfalldesign und zur Testfallgenerierung ein [115]. Testfälle, welche generiert werden, sind in erster Linie Modul- und Funktionstests für Embedded Systems, die immer eine abgeschlossene Softwarekomponente testen. Neben Conformiq gibt es noch weitere Firmen, die MBT fokussieren und dabei in Ansätzen auch Aktivitätsdiagramme als Modellierungsnotation zulassen.

Innovator Business Analyst (IBA) und MBTSuite

Die SeppMed[26] GmbH und die MID[27] GmbH bieten einen der ausgereiften Testfallgeneratoren (MBTSuite) an. Die MBTSuite ist in der Lage, mit Hilfe von Testprofilen in IBA modellierte Testprozesse zu interpretieren und daraus manuelle Testfälle zu generieren. Allerdings muss dazu ein spezielles Testprofil entwickelt werden, welches Testelemente beinhaltet sowie die BPMN und IBA erweitert. Gleichzeitig ist es in der Lage, Coderümpfe zu Testfällen zu generieren, welche dann geeignet ausprogrammiert werden müssen. Auch die offene Schnittstelle, welche das HPQC als Testmanagementtool bietet, nutzt die MBTSuite, um die generierten Testfälle zur Verwaltung und zur Dokumentation zu übertragen.

Neben dem HPQC werden auch Tools wie Rational Testmanager von IBM unterstützt. Allerdings ist die MBTSuite ein reiner Testfallgenerator und bietet keinerlei Modellierungsfunktionen. Deshalb muss zum Modellieren ein weiteres Werkzeug angebunden werden. Die MBTSuite wird bereits erfolgreich seit vielen Jahren in den verschiedensten Branchen zumeist in der Domäne der Embedded Systems eingesetzt. Untersuchungen, Erprobungen[28] oder auch Evaluierungen der MBTSuite in Bezug auf verteilte BIS gibt es bislang

[25] Activity Diagrams + Actions (generated from an Interface Diagram).
[26] http://www.seppmed.de/nc/en/startseite.html.
[27] http://www.mid.de/de/produkte/modellierungsplattform-innovator.html.
[28] Interview mit Axel Schadinsky und Martin Beisser von der SeppMed und MID GmbH.

nicht. Dafür unterstützt Seppmed bspw. beim Testen im Bereich der Medizintechnik, dort werden Abnahmetests von ophthalmologischen[29] Systemen mit dem sogenannten Innovator for Business Analyst[30] (IBA) modelliert (UML-Diagramme) und mit der MBTSuite können durch Integration Testfälle mittels IBA generiert werden. Ein ophthalmologisches System gehört zur Familie der ERP-Systeme und unterstützt bei der Diagnose von Krankheiten. IBA ist als klassisches Modellierungswerkzeug zu verstehen, welches nun nicht für die Geschäftsprozessmodellierung genutzt werden soll, sondern zum Modellieren von Testprozessen. IBA wurde nicht speziell für das Modellieren von Testfällen entworfen, sondern ist ein Modellierungswerkzeug, welches die BPMN zur Geschäftsprozessmodellierung im vollen Umfang anbietet. Deshalb soll im Rahmen dieser Dissertation erprobt werden, wie mit IBA Testfälle nicht in der UML, sondern in der BPMN modelliert werden können, gleichzeitig soll eine Lösung entwickelt werden, welche nur die BPMN-Elemente beinhalten soll, die auch tatsächlich für das Testen vonnöten sind.

All4Tec MaTeLo

All4Tec mit MaTeLo (Markov Test Logic) ist ein privat geführtes Unternehmen mit Sitz in Frankreich, wo es auch seinen größten Umsatz generiert. Sein Kerngeschäft ist das modellbasierte Testen, zu seinen schärfsten Mitbewerbern zählen Qtronic von Conformiq und der TestDesigner von TestDesign. All4Tec ist hauptsächlich (80% [30]) in der Automobilbranche tätig und bietet eine hoch entwickelte Software zum funktionalen Test von Blackbox-Verhalten. Delphi und PSA zählen zu den Referenzkunden der Firma. In Zukunft möchte man sich auch in die Medizinbranche einbringen und hier nach einen Absatzmarkt suchen. All4Tec gehört in die Klasse der modellbasierten Testfallgeneratoren [30]. Dieses Tool wird vorrangig beim Abnahmetest, sprich für die Validierung der Systemanforderungen bzw. der funktionalen Spezifikation, eingesetzt. Es kann sowohl für den Integrationstest als auch für den Regressionstest genutzt werden [75]. MaTeLo beinhaltet einen eigens entwickelten Editor zur Erstellung von Nutzungsmodellen, basierend auf Markov-Ketten. Markov-Ketten sind Zustandsdiagramme mit States und Transitionen, allerdings ergänzt um die Übergangswahrscheinlichkeit von einem Zustand zum nächsten sowie mit einem dedizierten Start- und Endpunkt [30], [33]. Der Nachteil ist die geringe Unterstützung von UML-Modellen bzw. die direkte Wiederverwendung von UML-Modellen, die in anderen Modellierungstools erstellt wurden. Aktuell ist eine professionelle Anbindung an Testmanagementtools noch nicht erfolgt, zumal MaTeLo nicht die Testdurchführung übernimmt, sondern allein Testfälle generiert, es besitzt also keine Ausführungseinheit. Ein Vorteil ist, dass das Anforderungsmanagement seit der Version 4.6 angebunden ist, zumal jetzt eine Anforderungsverfolgung bis zum Testfall implementiert wird. MaTeLo ist kompatibel mit allen üblichen Tools, wie IBM Doors© für das Requirements-Management und National Instruments TestStand© oder EXAM, der von Audi, VW und MicroNova gemeinsam entwickelten Testautomatisierung. Vor allem die Audi AG nutzt das Tool zur Modellierung ihrer Testfälle. All4Tec mit dem Tool MaTeLo ist auf das Testen von Embedded-Systemen spezialisiert, deshalb wird das Tool auch bei vielen Automobilherstellern (Renault, Peugeot, Citroën) sowie Autozulieferern wie Magneti Marelli usw. eingesetzt.

---

[29] http://www.seppmed.de/en/branchen/medical/ophthalmologie.html.
[30] Ein Produkt der MID-GmbH zur Modellierung von Geschäftsprozessen.

TestBench mit Tedeso

Die Imbus AG bietet für das modellbasierte Testen von Software die Testbench for Model-Based Testing (TestBench) mit Tedeso als Modellierungswerkzeug. Die Imbus AG bietet erst seit 2009 das MBT Tool (TestBench) an und sieht jetzt das modellbasierte Testen als Kerngeschäft ihrer Beratung [31]. Die Testbench stellt keinen eigenen UML-Editor zur Verfügung, d. h., Modelle können nur mit extern angebundenen Modellierungswerkzeugen erstellt werden. Dazu wird in der Praxis das Modellierungswerkzeug Tedeso genutzt. Der Testfallgenerator erzeugt auf Basis des C4-Überdeckungskriteriums (Pfadüberdeckung) zunächst abstrakte Testfälle, die noch spezifiziert werden müssen. Es umfasst den kompletten Sprachumfang der UML 2.0. Als Modellierungswerkzeug kann auch IBA genutzt werden. Als Alternative zu IBA bietet das Tool auch eine Schnittstelle zu Sparx Systems. Das Besondere hierbei ist, dass es damit möglich ist, UML-2-Profile zu definieren, um somit die Nutzung und die Mächtigkeit der UML auf die für die Testmodellierung relevanten Elemente einzuschränken. Der Vorteil dieses Werkzeugs ist die Integration in Testmanagementtools und die Ankopplung an verschiedene UML-Modellierungstools. Die Produkte von Imbus werden in den unterschiedlichsten Branchen eingesetzt, dazu zählen die Automobilbranche, das Gesundheitswesen und auch der öffentliche Bereich.

Smarttesting

Die französische Firma Smarttesting hat ihre Zentrale in Besancon und sieht sich selbst als Marktführer im Software-Testing-Marktsegment [30]. Der Umsatz wird etwa zur Hälfte aus dem Produkt- und Wartungsgeschäft erzielt und die andere Hälfte wird über Beratung und Projekte rund um das Testen generiert. Das Kerngeschäft von Smarttesting liegt in der Produktentwicklung [30], [31]. Der Testdesigner Smarttesting entstand aus einem Forschungsprojekt. Vormals in der Embedded-Domäne angesiedelt, driftet Smarttesting heute mehr und mehr in die BIS-Domäne [30]. Es ist integrierbar in führende Testmanagement- und Testautomatisierungswerkzeuge, insbesondere HPQC und Quicktest Professional. Der unmittelbare Mitbewerber von Smarttesting ist Conformiq mit Qtronic. Erfahrungswerte im Bereich der Enterprise-Applikationen gibt es bislang allerdings noch nicht [30], [31].

Rhapsody

Rhapsody, Automatic Test Generator (ATG) und TestConductor bilden eine durchgängige Werkzeugkette von der Analyse des Systems bis zur Testdurchführung. ATG und TestConductor integrieren sich in Rhapsody als Add-Inns und unterstützen ausschließlich Modelle, die mit diesem Werkzeug erstellt werden. Es gibt keine Integration zu führenden Testmanagementtools oder anderen Testwerkzeugen. Der TestConductor dient zur modellbasierten Entwicklung und zur Ausführung automatisierter Testfälle. Die generierten C++/C- oder Java-Testskripte können für Regressionstests eingesetzt werden [30], [131]. ATG unterstützt nur die Sprache C++, was gleichzeitig eines seiner zentralen Nachteile ist. Rhapsody wird in verschiedenen Branchen der Domäne Embedded Systems eingesetzt, vgl. [30], [131]. Sowohl die Autoren aus [30] als auch der Autor dieser Dissertation kommen zu der Erkenntnis, dass der Einsatz von TestConductor sich nur auf den Modultest beschränken lässt. Weiterhin verlangt der Einsatz von TestConductor eine intensive Ausei-

nandersetzung mit dem Sprachumfang von UML 2.0 und insbesondere auch mit UML-Testing-Profilen. Experten aus den jeweiligen Unternehmensfachbereichen werden sich nur sehr schwer tun, eine solche Modellierungssprache zu lernen, um dadurch eine bessere Testabdeckung zu erreichen. Auf die Vorstellung des Testwerkzeugs eXept der eXept Software AG wird im Rahmen dieser Dissertation verzichtet. Grund: Die bereits erläuterten Testwerkzeuge ähneln sehr dem Tool von eXept, weswegen auf [30] verwiesen wird.

2.4 Umfrage Modellierungstools und Testfallgeneratoren

Im Rahmen dieser Dissertation erfolgte eine an die jeweiligen Referenzkunden der MBT-Testwerkzeug-Anbieter gerichtete Umfrage, dabei sollte untersucht werden, in welcher Softwaredomäne z. B. MBT-Testwerkzeuge eingesetzt werden. Schon 2011 führten [138] eine generelle Umfrage zum Thema Testkultur in den verschiedenen Unternehmen durch. Dabei stellten die Autoren dieser Umfrage Fragen, die sich damit befassten, wie und nach welchen Vorgehensmodellen in den jeweiligen Unternehmen getestet wird, sowie Fragen zum Stellenwert des Testens in den Unternehmen. Neben den klassischen monumentalen und den agilen Vorgehensmodellen kristallisierte sich das W-Modell als ein Modell heraus, das in Unternehmen noch vor dem Wasserfallmodell Verwendung findet [138]. Im W-Modell erfolgt der Testprozess parallel zum Entwicklungsprozess und könnte bspw. bei Unternehmen mit einer hohen Anzahl an Mitarbeitern und mit einer zentralen Qualitätssicherung eine Rolle spielen. Für den Autor dieser Dissertation war das Fazit dieser Umfrage überraschend. Dieses stellt die Erhöhung der Leistungsfähigkeit vor die Senkung der Kosten, bspw. durch frühere Fehlerentdeckung mit Entwicklungs- und Testsystemen. Auch zeigte die Umfrage, wie enorm wichtig es ist, in neuen Testfeldern und über neue Testmethoden zu forschen. 56% der befragten Unternehmen gaben an, dass das Testende erreicht ist, wenn die Software ausgeliefert werden muss. Das Testen wird als eine Art Restaktivität am Ende der Kette gesehen [138]. Auch bewirkt der ständige Wettbewerbs- und Zeitdruck, dass das Testen vernachlässigt wird und somit bewusst Fehler in der Produktion in Kauf genommen werden. Dies steht im Widerspruch zu der Erhöhung der Leistungsfähigkeit. Das Ziel sollte es sein, den Testabdeckungsgrad zu erhöhen und dadurch Fehler früher im Entwicklungsprozess zu identifizieren, zu beheben und somit einen reibungslosen Ablauf in den Produktivsystemen zu gewährleisten.

Die Umfrage des Autors dieser Dissertation teilte sich in zwei Teile auf: Teil 1 fokussierte Fragestellungen bezüglich der Softwaredomäne, in der solche Tools eingesetzt werden, sowie die verwendete Modellierungsnotation und die Integrierbarkeit in führende Testmanagementwerkzeuge. Teil 2 geht der Frage nach den Auswirkungen und Veränderungen nach, die eine MBT-Methode generell mit sich bringt. Weil einer der befragten Referenzkunden darauf bestanden hat, nur an der Umfrage teilzunehmen, wenn nicht zurückzuverfolgen ist, welches Unternehmen welches Tool bezieht, fasse ich hier die Ergebnisse der Umfrage immer in Bezug auf alle Teilnehmer zusammen. Teilgenommen an der Umfrage haben vorwiegend die jeweils verantwortlichen Produkt-Testmanager. Diese Firmen besitzen zumeist Testteams oder dedizierte Testabteilungen, die aus 30 bis 50 Mitarbeitern bestehen. Der große Teil der befragten Referenzkunden setzt die MBT-Testwerkzeuge hauptsächlich im Bereich von Embedded Systems ein. An den Umfragen beteiligten sich: BMW, Daimler, Audi AG (VW), Delphi/PSA, Deutsche Telekom, Ericsson und Siemens

HealthCare. Leider gibt es momentan keine Unternehmen, die erfolgreich solche Testwerkzeuge im Bereich der verteilten BIS einsetzen, um diese auch zu interviewen.

2.4.1 Zusammenfassung der Ergebnisse von Fragenkatalog Teil I

In der Praxis herrschen Diskrepanzen in der Wahrnehmung zwischen den MBT-Testwerkzeug-Herstellern und den MBT-Kunden beim Einsatz von solchen MBT-Testwerkzeugen. Generell werben die Hersteller gerne mit dem Slogan: „Auf Knopfdruck Testfälle und Testskripte generieren." Dass das MBT nicht ganz einfach ist, stellen die Kunden dann spätestens beim Einsatz dieser Tools und bei der erstmaligen Modellierung von Testfällen fest.

Vor allem das Testen von medizinischen Diagnosegeräten erfolgt über eine Modellierung in Zustandsdiagramme. MBT-Testwerkzeuge werden auch in der Finanz- und Dienstleistungsbranche eingesetzt. Dort werden kleinere, kompakte Finanzprozesse modelliert und aus diesen werden dann Testfälle generiert. Die Entscheidungen für ein MBT-Testwerkzeug werden unter Zuhilfenahme von einfachen Entscheidungsbäumen [31] getroffen. Auch die Unternehmen, die sich nicht explizit auf die Entscheidungsbäume bezogen, nutzten intern ein ähnliches Entscheidungsmodell rein intuitiv, was vergleichbar mit [31] ist. Drei Viertel der acht befragten[31] Unternehmen besitzen ein lizenziertes Testmanagementwerkzeug. Davon setzt ca. die Hälfte ein Testmanagementtool als Ablage für generierte Testfälle ein. Die restlichen 25% bedienen sich etwa der Open-Source-Testmanagementtools oder einer eigenen, extra dafür entwickelten Datenbank. Hierzu gibt es nur wenige MBT-Testwerkzeuge, die eine Schnittstelle ohne hohen Aufwand unterstützen. Von den Interviewpartnern wurde bestätigt, dass das MBT-Testwerkzeug immer in Kombination mit anderen Testwerkzeugen benutzt wird. Die Bedienung erfolgt in den meisten Fällen von Entwicklern, diese werden durch Tester unterstützt. Als kritisches Feedback aus den Interviews kam, dass Tester zu unqualifiziert dazu sind, solche MBT-Testwerkzeuge zu bedienen, da ihnen grundlegende Modellierungskenntnisse fehlen. Auch können sie das generierte Testskript (Programmiercode) nicht immer nachvollziehen und verstehen. Bei den Modellierungsnotationen waren die Strategien unterschiedlich, es kommen sowohl UML, QML, Markov-Ketten als auch Petri-Netze zum Einsatz. Mit Aktivitätsdiagrammen werden Geschäftsprozesse für die Finanz- und die Dienstleistungsbranche modelliert, mit dem Hinweis, dass hier noch viel Lernbedarf besteht. Zusammenfassend lässt sich zu den eingesetzten Modellnotationen festhalten, dass sowohl die „Backgroundausbildung" der jeweiligen Testfallmodellierer als auch die Branche einen großen Einfluss darauf haben, welche Modellierungsnotation und welcher Modelltyp genutzt werden.

Die Daimler AG nutzt Testverfahren wie Hardware-in-the-Loop (HIL) Tests, bei denen Untersuchungen an Prüfständen und Fahrversuchen mit physikalischen Prototypen durchgeführt werden, sowie zusätzlich eine Methode zum Testen von Softwaremodulen mittels automatischer Testfallgenerierung. Dadurch wurden Testabdeckung und Testeffizienz erheblich verbessert. Ziel der Testautomatisierung ist ein umfassender Systemtest, bei dem möglichst viele relevante Systemzustände automatisch erreicht und überprüft werden. Zum

[31] Fragenkatalog Teil I befindet sich im Anhang dieser Dissertation.

Einsatz kommt das Conformiq Tool Qtronic. Hier werden mit Hilfe von Zustandsdiagrammen die verschiedenen Getriebezustände simuliert und mit Hilfe des Testmodells werden tausende Testfälle generiert. Ford und VW setzen zum Testen von Automobil-Steuergeräten Modena ein. Dabei ist das Modena-Konzept für das Kommunikationsverhalten von Steuergeräten konzipiert und soll das innere Verhalten von Steuergeräten prüfen. Da aber auch das Kommunikationsverhalten von Steuergeräten immer noch sehr komplex sein kann, muss mit Hilfe der UML-Zustandsdiagramme eine Abstraktion des Verhaltens modelliert werden. Der Vorteil ist die Wiederverwendbarkeit der Applikationsmodelle. Diese werden bei Veränderungen der Steuergerätesoftware entsprechend angepasst, aus dem neuen Modell können dann neue, angepasste Testfälle generiert werden.

Bei Ford werden die Fahrspurassistenten auch mit Modena getestet. Fahrspurassistenten sollen die Sicherheit im Straßenverkehr entscheidend verbessern, dadurch steigen die Anforderungen an die Tester immens. Testergebnisse können bis zum letzten Augenblick zu Veränderungen in der Software solcher Steuerungsgeräte führen. Je nach Komplexität wurden zwischen 1,5 bis 4 Personenmonate [117] zur Modellierung benötigt, wobei hier nur Neuanwender betrachtet werden. Bei den Modellen handelt es sich um Neuerstellungen. Ford musste feststellen, dass die Modellierer/Testdesigner eine bestimmte Vorbildung bezüglich Themen wie UML haben müssen, insbesondere beim Modellieren von Zustandsdiagrammen, welche mit Triggern usw. arbeiten.

Ericsson setzt beim Testen von Steuergeräten Qtronic ein. Dadurch konnte der Zeitaufwand für das Testen um ca. 70% reduziert werden [116]. Allerdings hat das MBT nicht nur Vorteile gebracht, die Modellierung der Testfälle ist aufwendig und erfordert eine besondere Expertise bei Testdesignern oder Testern generell. Die Vorteile liegen ganz klar in der Erhöhung der Testabdeckung. Außerdem können durch die veränderte Testmethodik Testfälle leichter gewartet oder neu generiert werden. Nachteilig sind die komplizierte Modellierungstechnik sowie der Initialaufwand.

2.4.2 Zusammenfassung der Ergebnisse von Fragenkatalog Teil II

Die erste Frage aus dem Fragenkatalog Teil II[32] bezog sich auf die Erwartungshaltungen der Kunden gegenüber dem eingekauften MBT-Testwerkzeug. Aufgrund der unterschiedlichen Bedienung der Testwerkzeuge unterscheiden sich ebenfalls die Antworten der Kunden. Zwei Drittel der Kunden von MBT-Testwerkzeugen haben den Initialaufwand für das Erstellen von Testmodellen unterschätzt. Die Erwartung der Kunden war es, Testfälle auf „Knopfdruck“, ohne erhöhten Aufwand bei der Modellierung, zu erhalten. Die Praxis zeigt, dass viele der im Einsatz befindlichen Testwerkzeuge kompliziert in der Bedienung sind und viele Informatikkenntnisse voraussetzen [23], [24], [25], [30]. Der Grund hierfür kann sein, dass viele dieser Testwerkzeuge im akademischen Umfeld entstanden sind. Der Vorteil ist, dass diese Testwerkzeuge zwar mit ihren ausgeklügelten Algorithmen Testfälle mit nahezu unendlichen Kombinationsmöglichkeiten generieren können, allerdings in einer kaum verständlichen Modellierungssprache. Diese Testwerkzeuge wären in Unternehmen kaum

[32] Fragenkatalog Teil II befindet sich im Anhang dieser Dissertation.

einsetzbar, eine direkte Zielgruppe wären Akademiker mit dem Schwerpunkt der Modellierung von Testfällen aus Anforderungen, wie die der Steuerung von Tachometern in Autos. Die Wahrnehmung des Autors dieser Dissertation ist, dass die Testwerkzeuge kaum mehr als Prototypen sind, die mit nur wenigen Schnittstellen zu anderen Testmanagementtools versehen sind. Auch die Erlernbarkeit befindet sich auf akademischem Niveau, bspw. bei den Algorithmen bei der Testfallgenerierung aus Modellen.

Hersteller solcher MBT-Testwerkzeuge gehen mit einer anderen Erwartungshaltung zum Kunden, denn sie gehen immer davon aus, dass die Kunden auch mit solchen komplexen Algorithmen umgehen können, ohne diese direkt darauf anzusprechen oder gegenüber den Unternehmen ehrlich zu sein und die Empfehlung auszusprechen, dass nur wenn der gesamte Softwareentwicklungsprozess modellbasiert erfolgt, ein modellbasierter Testansatz Aussicht auf Erfolg hat. Solche Voraussetzungen sind in der Unternehmenswelt kaum vorzufinden. Begründet durch die eher akademischen Ansätze bei der Entwicklung solcher Testwerkzeuge sind Unternehmen heute auch etwas zurückhaltender geworden, was Investitionen in Prototypen anbelangt. Viele der Testwerkzeuge können erst jetzt erste Referenzkunden vorweisen.

Integrationsfähigkeit in bestehende Testmanagementtools

Nur etwa die Hälfte (4) aller befragten Kunden von MBT-Testwerkzeug-Herstellern nutzen tatsächlich kommerzielle Testmanagementtools. Natürlich hat bei der Entscheidungsfindung das schon im Unternehmen eingesetzte Tool eine Rolle gespielt. Der andere Teil der Kunden nutzte entweder Open-Source-Testtools oder aber auch eigens entwickelte Fehlerdatenbanken und Testfallbeschreibungen in Excel. Die große Herausforderung ist es, die Kunden, die bisher kein Testmanagementtool im Einsatz haben, davon zu überzeugen, dass für den Erfolg des MBT auch das einzusetzende Testmanagementtool eine entscheidende Rolle spielt. Denn nur mit einem Testmanagementtool können die generierten Testfälle strukturiert und zentral für alle Beteiligten gespeichert werden. Auch bietet ein Testmanagementtool viel mehr Überwachungsmechanismen als eine Testdokumentation in bspw. Excel. Allerdings lassen sich Kunden nur sehr schwer von einem Testmanagementtool überzeugen und schlagen dem Testwerkzeug-Hersteller eher vor, eine passende Schnittstelle im schon vorhandenen Tool anzubieten oder diese zu entwickeln. Dies ließ viele Hersteller umdenken und eine ganzheitliche Lösung anbieten (wie Imbus). Sie integrieren die modellbasierten Techniken und Werkzeuge in ihre eigenen Testmanagement- und Testautomatisierungswerkzeuge. Andere hingegen ziehen sich nur auf die Testfallmodellierung (wie bspw. IBA) zurück und bieten die Möglichkeit der Anbindung, dafür entsteht zusätzlicher Aufwand bei der Implementierung solch einer Schnittstelle.

Anpassungs- und Beratungsaufwand

Das Testen wird durch das MBT nicht neu erfunden und mittlerweile gibt es einige Industriestandards, wie ISTQB Certified Tester [7]. Inzwischen werden in vielen Unternehmen Tester nach diesem Industriestandard zertifiziert. Damit das MBT eine Akzeptanz erfährt, muss es den Industriestandard-Prozess unterstützen. Bei der Frage nach den Kosten für Anpassung und Beratung hielten sich die Unternehmen zurück. So lässt sich aber zusammenfassen, dass der Beratungsaufwand bei Unternehmen mit einem etablierten Prozess im

Unternehmen (Softwareentwicklungs- und Testprozess) geringer einzuschätzen ist als bei Unternehmen, die nur bedingt nach einem Prozess arbeiten. Was als vorteilhaft gilt, ist, dass alle MBT-Testwerkzeuge sich im klassischen Testprozess unterordnen können. Aber für den Erfolg ist sicherlich ein eigenes Prozessmodell nützlich, das das Tool und die Methodik des MBT optimal unterstützt.

Auswirkungen von MBT auf Unternehmensorganisationen und auf die Vorgehensmodelle bei der Softwareentwicklung

Bei fast allen befragten Kunden gab es zu diesem Punkt keine ausführlichen Erläuterungen, bis auf die Tatsache, dass zwar das MBT im besten Falle den Entwicklungsprozess beeinflussen könnte, allerdings bisher Auswirkungen nicht wahrgenommen werden konnten. Auswirkungen gibt es bei den Testern, die sich jetzt auf neue Modellierungsmethoden und Modellierungssprachen einlassen müssen. Auch werden Tester früher in die Entwicklungsphase einbezogen. Bei keinem der befragten Unternehmen gab es allerdings eine klare Aussage darüber, ob die Tester evtl. ein Teil des Entwicklungsteams werden könnten.

Aus Sicht des Autors dieser Dissertation haben diese Unternehmen das MBT nur spärlich akzeptiert. Das Testen bekommt mit der Einführung von MBT als Testansatz eine neue Bedeutung im Unternehmen, denn das Testen wird an den Anfang des Entwicklungsprozesses verlagert. Allerdings spiegelt sich das bisher in keiner Unternehmensorganisation wider. Dies hat aber auch mit der Tatsache zu tun, dass viele der Werkzeuge tiefgehende Modellierungs- und Programmierkenntnisse erfordern und sich das Testen dadurch einfach viel mehr auf die Entwickler verlagert. Zumeist sind es ja tatsächlich auch sogenannte Whiteboxtests, die mit solchen Tools durchgeführt werden, daher ist auch der Ansatz, dass Entwickler diese Tests durchführen, nicht verkehrt.

Welche Auswirkungen hatte die Einführung auf die im Einsatz befindlichen Testprozesse und Teststufen in Ihrem Unternehmen?

Weil viele Unternehmen nach dem ISTQB-Standard arbeiten, waren keine größeren Auswirkungen festzustellen. Das liegt auch daran, dass die meisten der hier untersuchten Kunden die Testwerkzeuge im Bereich der eingebetteten Systeme einsetzen. Dort wird jetzt der sowieso immer durchgeführte Entwicklertest durch MBT-Testwerkzeuge unterstützt. Auswirkungen wären dann gegeben, wenn sich das Testen auf den Anwendertest übertragen lässt. Hier müsste jetzt ein Fachbereich zusammen mit einem zentralen Tester Testfallmodelle anhand der funktionalen Spezifikation entwerfen.

Wie ist die IT-Qualitätssicherung in Ihrem Unternehmen eingegliedert? Zentral? Dezentral (in den jeweiligen Entwicklungsabteilungen)?

Hier gab es unterschiedliche Angaben, von zentral bis dezentral und auch unorganisiert war alles dabei. Zu beobachten war allerdings, dass Unternehmen mit einem etablierten Testprozess entweder eine zentrale Testabteilung haben, die auch unabhängig agiert, oder ein Testteam war Bestandteil des Entwicklungsteams.

Zusammenfassend lässt sich aus den Eindrücken der Interviews Folgendes festhalten: Bei der Einführung von MBT in einem Unternehmen muss man realistisch bleiben. MBT

schafft Transparenz und die Erhöhung der Flexibilität beim Testen. Auch reduziert es die Testfalldefinition und die Testfalldurchführung. Eine eher unrealistische Eigenschaft ist, dass völlig neue Testfälle entstehen, die durch klassische Testansätze nicht zu erstellen wären. Auch dass das MBT einen etablierten Testprozess ablöst, ist eine Fehlannahme.

2.4.3 Beobachtete Seiteneffekte bei den Kundengesprächen

Mit der Einbindung von Testern bei der Definition der funktionalen Spezifikation (Anforderung) in der Softwareentwurfsphase werden einige Vorteile erzwungen [59], [61]. So können aus der funktionalen Anforderung nicht nur Testfälle modelliert werden, sondern in den Modellen können auch sogenannte Zusicherungen definiert werden. Allein aus den formulierten Zusicherungen lassen sich die ersten Testfälle spezifizieren. Diese bilden im automatisierten Modultest die Vor- bzw. Nachbedingungen und somit lassen sich Testfälle viel besser anwendungsspezifisch formulieren. Das Konzept wurde vom Urvater des „Design by Contract“[33], Bertrand Meyer, vorgeschlagen. Der große Vorteil von „Design by Contract“ ist, dass Entwickler und Tester frühzeitig beginnen, sich Gedanken über das Testen der zu entwickelnden Softwarekomponente und der Applikation zu machen. Aus den spezifizierten Vorbedingungen lassen sich Testfälle ableiten, was sich dann positiv auf das Testmodell auswirkt. Aus den Nachbedingungen, die gleichzeitig auch das Testorakel bilden, leitet sich das erwartete Systemverhalten ab. Dadurch können frühzeitig hochqualitative Testfälle durch Spezifizierung erstellt werden.

2.5 Modellbasiertes Testen in der Domäne der verteilten betrieblichen Informationssysteme

Zur Umsetzung von Unternehmensstrategien und Geschäftsanforderungen benötigen Unternehmen die Fähigkeit, Geschäftsprozesse und Innovationen von betriebswirtschaftlichen Konzepten schnell in eine Softwarelösung zu übertragen. Diese Softwarelösungen agieren meist in einer verteilten heterogenen Systemlandschaft und müssen mit verschiedenen Applikationen kommunizieren. Verteilte BIS zeichnen sich dadurch aus, dass an ihnen immer mehrere Personen mit verschiedenen Rollen und Verantwortlichkeiten arbeiten. Meistens besitzen sie eine komplexe Architektur und sind workflow-gesteuert. Im Gegensatz zu Embedded Systems werden verteilte BIS von Geschäftsprozessen getrieben und haben mehrere Schnittstellen zu angebundenen Systemen. Beim Testen solcher Systeme werden die Schwerpunkte auf kritische Workflows, Geschäftsprozesse und Geschäftsprozessvarianten gelegt. Ein Schritt, um die Komplexität beherrschbarer zu machen, ist die Modellierung dieser Geschäftsprozesse. Allerdings nimmt die Modellierung keine Komplexität aus den Applikationen, sondern erlaubt nur einen etwas abstrakteren Blick auf die Geschäftsprozesse. Getreu dem Motto: „Ein Bild sagt mehr als 1000 Worte.“

[33] Die Idee des Contracts zwischen Client und Server stammt aus dem Geschäftsleben: Zuerst erklärt sich der Kunde bereit, dem Dienstleister eine gewisse Summe zu zahlen (entspricht Obligation des Clients). Der Dienstleister wiederum verpflichtet sich, dem Kunden dafür etwas Bestimmtes zu liefern (entspricht Obligation des Servers). Dieses Schema wird nun auf den Software-Engineering-Prozess übertragen.

2.5.1 MBT-Ansätze in betrieblichen Informationssystemen

Handelt es sich bei Embedded Systems meistens um Ingenieure und technisch versierte Tester, die auch in der Lage sein müssen, z. B. UML-Diagramme zu verstehen und selbst zu modellieren, sind es in der Domäne der BIS oft Mitarbeiter aus einem speziellen Fachbereich, die meistens weder technisch noch programmiertechnisch versiert sind. Allerdings verfügen diese Personen über große fachliche Kenntnisse in der Prozesssicht und in der Geschäftsprozesslogik von BIS. Bedingt durch die hohen Abhängigkeiten verteilter BIS gestaltet sich die Validierung dieser Systeme schwierig. Die hohen Abhängigkeiten erfordern einen aufwendigen Test [78]. Um diesem entgegenzutreten, wird versucht, das, was aus heutiger Sicht in der Domäne der Embedded Systems gängige Praxis ist [12], [13], auf die Domäne der BIS zu übertragen. Deshalb ist MBT mittlerweile auch Forschungsgegenstand in der Domäne der BIS. In [13] wird am Beispiel von SAP gezeigt, wie ein Testfall per UML auf der GUI-Ebene modelliert werden kann; [14] beschreibt einen theoretischen Ansatz, wie ERP-Systeme generell modellbasiert (UML) verifiziert und validiert werden können. Zusammengefasst beinhalten die Ausarbeitungen gute theoretische Ansätze, auf denen eine MBT-Methode für verteilte BIS aufgebaut werden kann [19].

Die MID[34] GmbH bietet seit September 2012 eine neue Modellierungsplattform, IBA, an. Diese erlaubt die Geschäftsprozessmodellierung komplexer Geschäftsprozessabläufe eines Unternehmens. Dazu werden die aus der objektorientierten und strukturierten Softwareanalyse [64] bekannten Diagramme genutzt. Eingesetzt wird dieses Tool vermehrt in der modellgetriebenen Softwareentwicklung [118]. IBA ist im Gegensatz zu den schon genannten Werkzeugen ein reines Modellierungstool für Geschäftsprozesse und unterstützt die gängigen Industriestandards wie die BPMN und die UML vollständig. Zusätzlich muss jetzt dieses in erster Linie für Anforderungsanalysten und Prozessdesigner konzipierte Werkzeug umfunktioniert werden, damit vermehrt Tester (Business-Analysten), welche Modellierungen von Testprozessen übernehmen, dieses Werkzeug nutzen.

IBA und die Integration der MBTSuite

Das Testmodell kann mit IBA [121] angefertigt werden, zusätzlich werden ein Testwerkzeug und eine Erweiterung von BPMN benötigt, um aus dem modellierten BPMN-Modell Testfälle abzuleiten. Über eine IBA-Schnittstelle kann die MBTSuite angebunden werden. Nachteil: Es müssen zwei verschiedene Werkzeuge (das Modellierungswerkzeug und der Testfallgenerator) miteinander agieren. An dieser Stelle entstehen doppelte Lizenzkosten und auch die Schnittstelle zwischen IBA und MBTSuite kann eine Fehlerquelle darstellen, zumal die Konfigurationen und das Erstellen von Profilen sich nicht als „einfach“ erweisen. Das Generieren von Testskripten aus manuellen Testfällen ist mit der Kombination von IBA und MBTSuite ohne spezielle Anpassungen (Adaptierungen) nicht möglich. Die mittels BPMN generierten Testcoderümpfe müssen geeignet implementiert oder eine Referenz muss eingebaut werden, welche auf fertig implementierte Testbausteine in Bibliotheken verweist. Der große Vorteil von IBA liegt in der Modellierung in BPMN und in der Tatsache, dass durch Erweiterungen der Spezifikation, durch Konfigurationen und durch das Erstellen von Testprofilen[35] ein

[34] http://www.mid.de/de/produkte/modellierungsplattform-innovator.html.
[35] Wird im Kernteil (Kapitel 3 und 4) genauer vorgestellt.

„Missbrauch“ als Testmodellierungstool möglich ist. Gleichzeitig kann ein Testprofil angefertigt werden, welches auch übertragbar ist. So kann ein einmalig erstelltes Profil dazu dienen, sämtliche Testprozesse in BPMN abzubilden. Auch kann es in verschiedenen Unternehmen eingesetzt werden. Das Profil ist unabhängig vom eingesetzten Unternehmen, da es keine unternehmensspezifischen Informationen beinhaltet, sondern sich auf die Testspezifikation konzentriert. Die Erweiterungen der generierten Testfälle zu fertig ausführbaren Testskripten machen Adaptierungen nötig, welche im Kernteil dieser Dissertation näher betrachtet werden.

- Folgende Forschungsfragen müssen im Rahmen dieser Dissertation noch näher untersucht werden:
- Lassen sich mit IBA und MBTSuite integrative Geschäftsprozesse als Testfall in BPMN modellieren und lassen sich diese mit Klassendiagrammen spezifizieren?
- Wie müssen die generierten Testfälle adaptiert werden, damit sie zu automatisierten Testskripten werden? Hier muss eine Adaptierung entworfen und zumindest gezeigt werden, wie die Testfälle auf Testskripte abgebildet werden und wie dadurch ausführbare Testskripte entstehen.
- Wie können schon vorhandene Programmierbibliotheken (von Testskripten) integriert werden?

Funktionsbibliotheken müssen eingebunden werden, damit das, was jetzt schon vorhanden ist, nicht verloren geht.

Conformiq

Conformiq versucht, mit RUDI eine einheitliche Lösung auf den Markt zu bringen. Das bisherige Vorgehen ist, dass das Modellieren von Testfällen in Form eines Zustandsdiagrammes in QML erfolgt. Daraus werden dann Testfälle generiert, die wiederum angepasst und bearbeitet werden können. Im nächsten Schritt können aus den Testfällen mit Hilfe der Entwicklungsumgebung von Conformiq Testskripte erstellt werden. Mit RUDI will Conformiq das Modellieren mit Aktivitätsdiagrammen analog zu Zustandsdiagrammen ermöglichen. Die Aktivitätsdiagramme beschreiben die Ablauflogik eines Geschäftsprozesses. Zusätzlich können Schnittstellen mit Informationen zu anderen Systemen modelliert werden. Es erlaubt aber keine Spezifizierung von Testdaten zu den Schnittstellen, die über ein „Ja“ oder „Nein“ hinausgehen. Auch ist die QML als Notationssprache zwischen UML und BPMN eine sehr schwer zu erlernende Notation, welche sich gezielt an IT-Experten wendet. Der Vorteil dieser Lösung ist die Integration in gängige Testmanagementtools wie HP ALM.

Zusammenfassend lässt sich an dieser Stelle festhalten, dass die MBTSuite mit IBA für die Evaluierung und die Erprobung der in dieser Dissertation formulierten Anforderungen genutzt werden soll. Nicht nur, dass sich BPMN als Modellierungsnotation anbietet, sondern auch das Einbinden der Fachbereiche erfordert eine Modellierungsnotation, die näher an den Anwender einer Software angelehnt ist als an einen IT-Experten oder Entwickler. Auch die Untersuchung und Evaluierung durch den Autor dieser Dissertation brachte die Erkenntnis, sich auf die MBTSuite und IBA zu konzentrieren und die dort notwendigen Erweiterungen und Anpassungen dann im Kernteil (Kapitel 3 und 4) im Detail vorzustellen.

2.5.2 Voraussetzungen für das MBT in verteilten BIS

Die Voraussetzungen für einen Einsatz von MBT in verteilten BIS sind vielfältig und in Ansätzen noch genauer zu untersuchen [17]. Die Basis kann aber aus etablierten Ansätzen, wie dem Embedded-System, gewählt werden. Zusammenfassend lassen sich die Voraussetzungen wie folgt formulieren:

- Es muss ein standardisierter Testprozess in einer Unternehmensorganisation existieren, bevor über MBT nachgedacht wird.
- Ein Template für funktionale Spezifikationen, die die allgemeine Gültigkeit dieser Dokumente sicherstellen, muss existieren. D. h., es muss zumindest ein statischer Test (Review der Dokumente) stattgefunden haben.
- Qualifizierte Mitarbeiter sind notwendig, die sich nicht nur mit den korrekten Formulierungen von Anforderungen in funktionale Spezifikationen auskennen, sondern auch mit der Modellierung solcher Anforderungen in Form von Testfällen.
- Sowohl eine Testsystemlandschaft als auch Testwerkzeuge und Modellierungswerkzeuge müssen vorhanden sein oder zumindest im Rahmen einer Einführung von MBT integriert werden.

Ein Geschäftsprozess umfasst eine Menge von manuellen und automatisierten betrieblichen Aktivitäten, die nach bestimmten Regeln oder Vorgaben auf ein bestimmtes Ziel hin ausgeführt werden. Ein Geschäftsprozess wird dann ausgelöst, wenn er einem Kunden einen Nutzen erbringt und gleichzeitig Unternehmen die Administration der betrieblichen Abläufe erleichtert. Um sowohl die funktionalen Spezifikationen besser interpretieren zu können als auch die Testfallmodellierung zu erleichtern, können Anwendungsfälle herangezogen werden [6]. Diese bilden dann die Grundlage für die Modellierung von Testfällen auf der Basis der funktionalen Spezifikation.

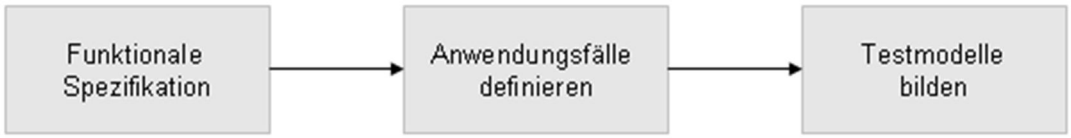


Abbildung 10: Von der Anforderung zum Testmodell

Die spezifizierten Anwendungsfälle erleichtern das Verständnis, wie ein Softwaresystem funktionieren soll. Prof. Schlingloff und Mario Friske beschreiben und illustrieren in [43] eindrucksvoll, wie man systematisch von einem Anwendungsfall zu einem Testfall gelangen kann. Die dort ausgearbeiteten Erkenntnisse und Ergebnisse können jetzt genutzt werden.

2.6 Themenbezogene Arbeiten

Das Thema des modellbasierten Testens unter Anwendung von Softwaremodellen, insbesondere von UML-Modellen [88], wurde in der Vergangenheit bereits vielfach diskutiert [16]. Zahlreiche Arbeiten sind zu dem Schluss gekommen, dass die automatische Generierung von Testszenarien, Testfällen und Testplänen, basierend auf den im Systementwurf entstehenden UML-Modellen, durchaus machbar und auch sinnvoll ist [21]. In [21] werden formale Grundlagen zur Generierung von Testfällen aus UML-Zustandsdiagrammen gelegt. Weiterhin wird auf dieser Grundlage ein Algorithmus zur Erzeugung von Testfällen

aus UML-Zustandsdiagrammen vorgestellt. Dabei kommen [23] und [24] in ihren Arbeiten zu dem Schluss, dass die traditionellen Whiteboxmethoden zur Erzeugung von Testfällen aus funktionalen Spezifikationen besonders bei komplexen, wie objektorientierten, Systemen nicht geeignet sind.

Zum einen erreichen sie nur eine ungenügende Zustandsabdeckung, zum anderen gestaltet sich die automatische Testfallgenerierung kompliziert. Dies ist darin begründet, dass ausschlaggebende Aspekte des Systemdesigns schwer aus dem Quellcode extrahiert werden können [23].

Unter der Annahme, dass sich der Zustand eines Systems zu jedem Zeitpunkt eindeutig durch die Zustände seiner Komponenten definiert, wird in [24] eine Methode vorgeschlagen, um eine adäquate Abdeckung der Systemzustände für Testfälle zu erreichen. Hierzu werden UML-Anwendungsfalldiagramme, UML-Zustandsdiagramme sowie UML-Sequenzdiagramme analysiert und für Sequenzen von Anwendungsfällen erreichbare Systemzustände berechnet. Aus einer Menge solcher Sequenzen von Anwendungsfällen wird eine geeignete minimale Auswahl getroffen, die gemessen an einem zuvor festgelegten Grenzwert eine hinreichend große Systemabdeckung erreicht [93]. In [23] wird ein Ansatz zur Generierung von Testfällen aus UML-Anwendungsfalldiagrammen und UML-Sequenzdiagrammen vorgestellt.

Einen quantifizierenden Vergleich von verschiedenen UML-Diagrammtypen zur automatisierten Testfallgenerierung stellen Kansomkeat, Offut et al. in [27] vor. Anhand eines akademischen Beispiels werden die Fehlererkennungsraten von aus UML-Sequenzdiagrammen und UML-Zustandsdiagrammen generierten Testfällen miteinander verglichen. Im Ergebnis steht die Aussage, dass UML-Modelle prinzipiell zur Generierung von Testfällen geeignet sind. Unterschiede ergeben sich jedoch bei der Eignung solcher Testfälle je nach Art des UML-Diagrammtyps und je nach Art des durchgeführten Tests (Modultest oder Integrationstest). So eignen sich für elektronische Steuergeräte Zustandsdiagramme besser als Aktivitätsdiagramme. Und umgekehrt eignen sich Aktivitätsdiagramme besser zur Modellierung von Geschäftsprozessabläufen als Zustandsdiagramme.

Samuel und Joseph präsentieren in [26] einen weiteren Ansatz zur Generierung von Testfällen aus UML-Sequenzdiagrammen. Es liegt die Annahme zugrunde, dass UML-Sequenzdiagramme Realisierungen von Anwendungsfällen repräsentieren, welche letztendlich möglichen Testszenarios entsprechen. Es wird ein Algorithmus vorgestellt, welcher Sequenzen von Nachrichten zwischen Objekten im UML-Sequenzdiagramm analysiert. Diese werden zu Sequenzen von Nachrichten gruppiert, welche in Relation zueinander stehen. Ähnlich wie das Konzept aus [27] wird auch hier das UML-Diagramm in eine intermediäre Form der Darstellung umgewandelt, den „Sequence Dependency Graph“, aus welchem schließlich Testfälle extrahiert werden. Man kann hier jede Sequenz im Sequenzdiagramm als einen Testfall sehen, mit dem Nachteil, dass dadurch eine Inflation an Testfällen generiert werden kann. Ein Nachteil ist dies, weil die generierten Testfälle Abhängigkeiten zu anderen Testfällen haben könnten und verwaltet werden müssen. Auch müssen die generierten Testfälle einem Dispatcher oder Scheduler zur Verarbeitung übergeben werden. Eine Flut an Testfällen ist hier nicht förderlich, allerdings für die Ausführung durch einen Automaten unerheblich.

Während die oben genannten Arbeiten die Generierung von Testfällen aus UML-Zustandsdiagrammen, UML-Sequenzdiagrammen oder Anwendungsfalldiagrammen zum Thema haben, wird in [99] der Erzeugung von Testszenarien aus UML-Aktivitätsdiagrammen der Vorzug gegeben. Es wird ein theoretischer Ansatz vorgestellt, welcher in der Praxis bisher nicht erprobt wurde und daher etwas vage ist.

Ein weiterer Ansatz zur Generierung von Testfällen aus UML-Aktivitätsdiagrammen wird in [28] vorgestellt. Hier wird ebenfalls eine durch Modelltransformation erzeugte Form der Darstellung des UML-Aktivitätsdiagramms benutzt, um Testfälle zu generieren. Im Gegensatz zu [99] und [100] wird jedoch das Konzept verfolgt, Eingaben in das System und Ausgaben aus dem System explizit darzustellen. Aber auch die Akteursaktionen und die Systemaktionen werden bei der Modellierung unterschieden. So wird sichergestellt, dass nur diejenigen Aktionen bei der Testfallgenerierung berücksichtigt werden, die tatsächlich für einen Tester erreichbar sind. Um eine hinreichend große Testabdeckung des SUT[36] zu gewährleisten, wird ein Abdeckungskriterium eingeführt. Dieses stellt einerseits sicher, dass alle testrelevanten Aktionen getestet werden, aber andererseits nicht unnötig viele redundante Testfälle erzeugt werden. Obwohl die Definitionen von Whitebox-, Greybox- und Blackboxtests[37] in den zuvor genannten Arbeiten teilweise differieren, haben sie dennoch gemeinsam, dass Testfälle generell aus modellierten Systemspezifikationen automatisch generiert werden können.

Die Ansätze zur Testfallgenerierung sind unterschiedlich und es kommen verschiedene UML-Diagrammtypen zum Einsatz. Die Überführung eines UML-Diagramms in eine an die besonderen Anforderungen angepasste Form der Darstellung ist eine etablierte Praxis [109], [110]. Einerseits werden so die besonders relevanten Aspekte für einen Testfall in den Vordergrund gestellt, andererseits kann die Komplexität der Testfallerstellung aus UML-Diagrammen unter Kontrolle gebracht werden. Unter Umständen werden Verarbeitungsschritte in einem Softwaresystem vollständig automatisch ohne Benutzerinteraktion durchgeführt. Sie entziehen sich somit dem direkten Zugriff durch einen Tester und sollten aus diesem Grund nicht Bestandteil eines Testfalls sein [111].

Diese Arbeit wird die automatisierte Testfallgenerierung um weitere Modelle erweitern. Durch die Kombination von UML-Modellen mit der Modellierungssprache BPMN 2.0 wird versucht, die Testfallgenerierung nicht nur für den Whiteboxtest zu erstellen, sondern auch für fachliche Blackboxtests, die das Testen eines Geschäftsprozesses in den Vordergrund stellen.

Mehrere Arbeiten, darunter auch [23], [24], kommen zu dem Schluss, dass die UML-basierten Testfallgenerierungsansätze zwar fortgeschritten sind, sich diese aber zumeist auf die Domäne der Embedded Systems beschränken und dort auch Erfolg haben. Was jetzt untersucht werden muss, ist, wie bestehende Ansätze genutzt werden können, um die erfolgreichen Aspekte aus den diversen Ansätzen auch auf eine weitere Domäne zu übertragen. In dieser Domäne dominieren Aktivitätsdiagramme, da diese Diagramme für Fachbereiche, die kein Ingenieur- oder Informatikwissen vorweisen können, besser und verständlicher zu lesen sind. BPMN bietet fortgeschrittene und erweiterte Aktivitätsdiagramme, die einem Betrachter auf einen Blick den Ablauf eines Geschäftsprozesses erklären.

---

[36] System Under Test.

[37] Eine genaue Definition der Begriffe bieten: [7], [10].

2.7 Was ist BPMN?

Dieser Abschnitt gibt einen kleinen Einblick in die Hintergründe von BPMN [105] sowie in die für diese Dissertation relevanten BPMN-Elemente[38] und die notwendigen Erweiterungen von Stereotypen zur Testmodellierung in BPMN. BPMN ist der Standard für die grafische Beschreibung von Geschäftsprozessen und deren Abhängigkeiten untereinander. Es ist eine Spezifikationssprache für die Modellierung und die Dokumentation von Geschäftsprozessen und Arbeitsabläufen.

Zielgruppen von BPMN-Modellen sind einerseits die Business-Anwender im Fachbereich [32], die in der Lage sein sollen, die BPMN-Diagramme zu erstellen und zu lesen, und andererseits die Softwareentwickler (Implementierer), die diese Diagramme in eine konkrete Implementierung transferieren sollen. Ein Business-Analyst kann dann zusammen mit den Anwendern im Fachbereich und mit den Entwicklern Testmodelle auf der Basis von BPMN erstellen. Auf diese Weise entsteht eine standardisierte Brücke zwischen diesen drei Benutzergruppen und Rollen.

2.7.1 Entstehung von BPMN

Eine Chronologie der Entstehung:[39]

- ✓ 2002: erste BPMN-Spezifikation / Stephen White (IBM)
- ✓ 2004: BPMN wird von der Business Process Management Initiative (BPMI) veröffentlicht
- ✓ 2005/2006: Fusion von BPMI und Object Management Group (OMG)[40]
- ✓ seit 2006: BPMN als OMG-Standard (neben UML etc.)

Angestoßen wurde BPMN vom ehemaligen IBM-Mitarbeiter Stephen White. Die Veröffentlichung erfolgte 2004 durch die Organisation Business Process Management Initiative (BPMI)[41] mit Stephen White als Autor der Spezifikation. Daraufhin gründete die BPMI eine kombinierte Gruppe zur Weiterentwicklung von BPMN unter der Bezeichnung Modeling and Integration Domain Task Force. Ab diesem Zeitpunkt befand sich die BPMN-Entwicklung unter dem OMG-Dach. BPMN wurde ursprünglich für Prozessbeschreibungen entwickelt, die bspw. dann von einem Prozessmanagementsystem ausgeführt werden können. Allerdings lassen sich mit BPMN 2.0 generell Prozesse modellieren und dadurch implizit auch Testprozesse [94], [95]. BPMN erfreut sich immer größerer Beliebtheit, vor allem in Unternehmensfachbereichen [151]. Die Testfallmodellierung lässt sich mit Hilfe von BPMN leicht und leserlich für Fachbereiche erstellen [89], [90].

[38] [105] enthält eine ausführliche Einführung in BPMN und informiert über alle in BPMN verfügbaren Elemente.
[39] http://bpmn.org/Documents/BPMN_V10_May_3_2004.pdf.
[40] http://www.omg.org.
[41] http://www.bpmi.org/.

2.7.2 BPMN-Basiselemente

BPMN als grafische Anwendung definiert zwar einen überschaubaren Wortschatz, wodurch es einfach zu verstehen und anzuwenden ist. Die Spezifikation definiert insgesamt vier Gruppen von BPMN-Elementen [151], welches gleichzeitig die Basiselemente dieser Notation sind. Dazu gehören Aktivitäten, welche Handlungen oder Arbeitsschritte definieren, Sequenzflüsse, die den Verlauf eines Prozesses anzeigen, Ereignisse, welche mit Zeitintervallen und Funktionalitäten verbunden sind, und Gateways, mit denen Verzweigungspunkte spezifiziert werden können. Mit Hilfe von Artefakten können zu den Elementen zusätzliche Informationen hinterlegt werden. Für die Testfallmodellierung bieten sich vor allem Datenobjekte an, in denen Testdaten hinterlegt werden können. Ein Datenobjekt kann wiederum eine externe Datei (bspw. Excel oder UML-Klassendiagramme) bedingen.

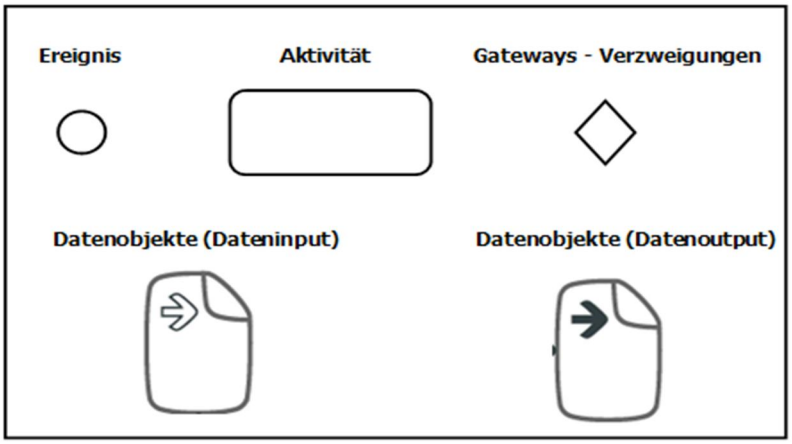


Abbildung 11: BPMN-Basiselemente

Zu den Verbindungsobjekten zählen Sequenzflüsse, Nachrichtenflüsse und Assoziationen. Zu den Artefakten zählen Datenobjekte, Kommentare sowie Gruppen und zu den Schwimmbahnen Pools und Lanes. All diese Elemente werden im weiteren Verlauf für die Modellierung des Beispielprozesses dieser Dissertation herangezogen. So können z. B. Zuständigkeiten für Aktivitäten definiert werden, indem ein Prozessteilnehmer, wie z. B. eine Organisation im Unternehmen oder auch eine Abteilung, jeweils als Pool dargestellt und die ausführenden Einheiten jeweils als eine eigene Bahn (Lane) im Pool definiert werden. Ein Prozessteilnehmer kann aber auch ein Softwaresystem sein, welches Aktivitäten im Geschäftsprozess übernimmt. Sowohl Gateways als auch Ereignisse bestehen aus mehreren Elementen, siehe obige Abbildung.

2.7.3 Schwimmbahnen

Prozessteilnehmer werden in Form von Schwimmbahnen dargestellt, wobei ein Pool aus mehreren Schwimmbahnen bestehen kann.

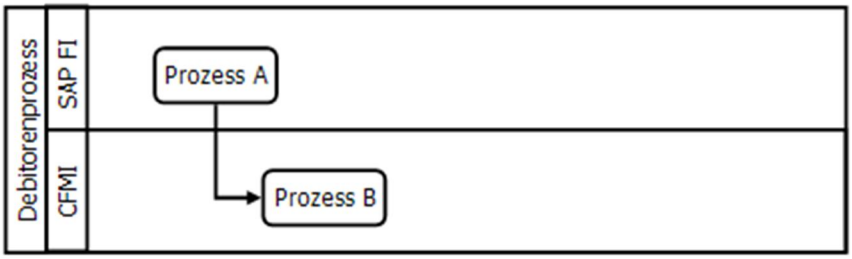


Abbildung 12: BPMN Swimlanes (Schwimmbahnen)

Ein Pool oder mehrere Schwimmbahnen repräsentieren Verantwortlichkeiten für Aktivitäten und dies kann eine Organisation, eine Rolle oder ein Softwaresystem (Applikation) sein. Pools und Lanes können sowohl im aufgeklappten als auch im zugeklappten Zustand dargestellt werden. In einen Geschäftsprozess involvierte Teilnehmer können durch Pools separiert werden. Innerhalb eines Pools werden wiederum jeweils eigene Prozesse definiert. Die Darstellung des zugeklappten Zustands wird benötigt, wenn Verbindungen zwischen einzelnen, miteinander agierenden Systemen gezeigt werden sollen, ohne dabei konkret auf den internen Aufbau dieser Systeme einzugehen. Ist z. B. der Geschäftsprozess eines Geschäftspartners nicht bekannt, kann dieser in einem zugeklappten Pool versteckt werden. Die Kommunikation zwischen zwei Pools (Systemen) kann mittels Nachrichtenfluss dargestellt werden. In der Grafik steht CFM für Customer File Managementsystem und SAP-FI für die Buchhaltungssoftware der SAP AG. CFM-Systeme werden auch synonym CRM-Systeme genannt. Sowohl Oracle mit Siebel als auch die Sage Software GmbH gelten als führend im Vertrieb dieser Softwarelösungen.

2.7.4 Fluss-Objekte

Fluss-Objekte eignen sich dazu, das Verhalten eines Geschäftsprozesses grafisch zu modellieren, dabei werden die Fluss-Objekte in folgende drei Gruppen unterteilt [151]:

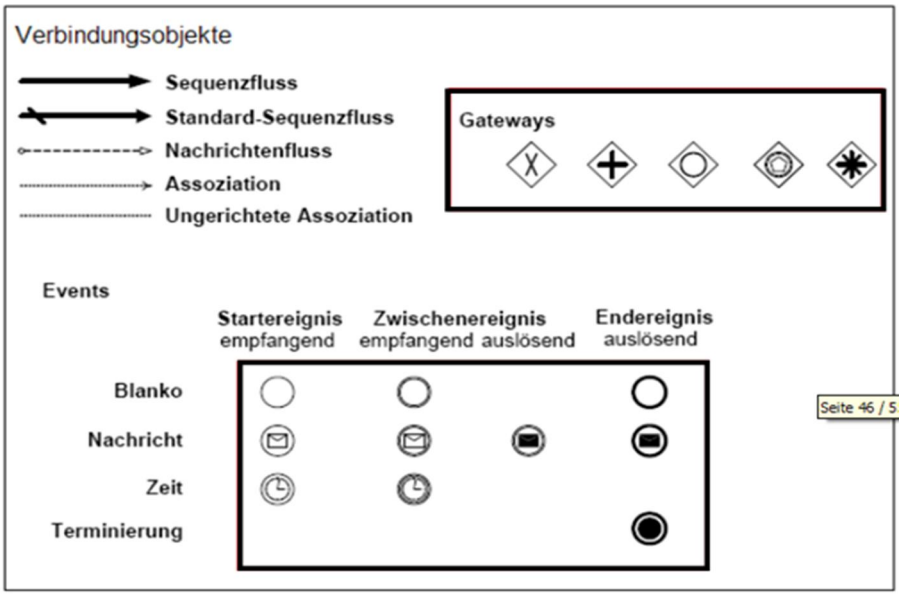


Abbildung 13: Verbindungsobjekte und Eventsymbole

Events modellieren Ereignisse, die in Prozessen oder Choreographien auftreten. Die Ereignisse haben entweder eine Ursache („trigger“) oder eine Wirkung („result“). Dargestellt wird ein Event durch einen Kreis, in dessen Mitte ein Symbol für die Art des Events platziert wird. Klassifiziert werden Ereignisse nach dem Auftreten innerhalb des Prozesses: Start-, Intermediate- und End-Event. Eine Aktivität („activity“), wozu auch Aufgaben, Transaktionen, Ereignis-Teilprozesse und Aufruf-Aktivitäten gehören, wird als Rechteck dargestellt.

Rechtecke beschreiben Aufgaben, die innerhalb eines Geschäftsprozesses zu erledigen sind, wie z. B. „Kasse abschließen“.

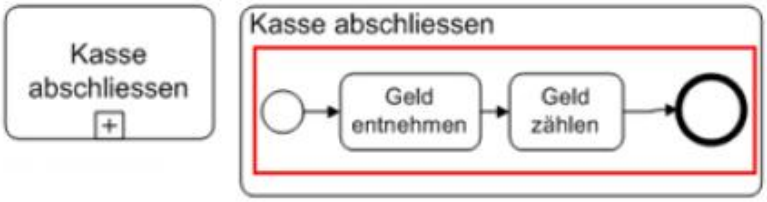


Abbildung 14: Kassenabschluss als Prozess zusammengefasst

Eine Aufgabe kann durch einen eigenen Prozess genauer spezifiziert werden, so können Prozesse wiederum Unterprozesse (Teilprozesse) enthalten. Diese werden durch ein kleines „Pluszeichen“ in der Mitte des Symbols repräsentiert.

Durch die Verwendung von Gateways werden bei den Prozessabläufen Verzweigungen von Geschäftsprozessen dargestellt [105], [151]. Zusätzlich können Zusammenführungen von mehreren Prozessausführungen dargestellt werden. BPMN bietet fünf Arten von Gateways (default, AND, OR, XOR und ereignis-basierte Gateways). Ein „default“-Gateway ist per Spezifikation ein XOR-Gateway.

2.7.5 Datenobjekte

Data spezifiziert die Daten, die während der Ausführung eines Geschäftsprozesses als Eingabe benötigt und als Ausgabe geschrieben werden. So können auch Äquivalenzklassen gebildet werden, um eine Kombination von Eingabemöglichkeiten zu realisieren. Die einzelnen Elemente werden durch Assoziationen mit Fluss-Objekten oder mit einem Sequenzfluss verknüpft, wobei eine Assoziation die Verbindung mit einem anderen Element beschreibt und ein Sequenzfluss den Kontrollfluss innerhalb des Prozesses modelliert.

Ein Datenobjekt stellt zum einen die Informationen bereit, die eine Aktivität (Aufgabe) für ihre Ausführung benötigt. Zum anderen werden durch Datenobjekte Informationen gespeichert, die von einer Aktivität während ihrer Ausführung erzeugt werden. Dabei kann entweder ein einzelnes Objekt oder eine ganze Sammlung von Objekten, eine sogenannte Collection, repräsentiert werden [152].

Mit Dateninputs und Datenoutputs können die Ein- und Ausgabedaten eines Prozesses definiert werden. Die Daten können von einzelnen Aktivitäten gelesen und geschrieben werden. Die Dateninputs werden durch einen nicht ausgefüllten Pfeil, die Datenoutputs durch einen ausgefüllten Pfeil dargestellt (Abbildung 11). Ein zusätzliches Objekt ist das sogenannte Datenspeicherobjekt. Dieses stellt einen Mechanismus bereit, um Daten, die über die Laufzeit des Prozesses hinaus existieren, zu lesen und zu schreiben.

2.7.6 Verbindungsobjekte

Verbindungsobjekte sind die verbindenden Kanten zur Darstellung des Kontroll- oder Datenflusses sowie zur Verbindung von weiteren Elementen innerhalb eines Geschäftsprozesses. Es existieren vier Arten von Verbindungsobjekten: Sequenzfluss, Nachrichtenfluss, Assoziation, Standard-Sequenzfluss.

Ein Sequenzfluss verbindet zwei Flow Objects und stellt die Ausführungsreihenfolge dar. Dargestellt wird er durch einen Pfeil mit durchgezogenem Strich und ausgefülltem Pfeilende. Ein Nachrichtenfluss zeigt den Nachrichtenaustausch zwischen zwei Elementen innerhalb eines Geschäftsprozesses an [105]. Er wird als gestrichelter Pfeil mit einem nicht ausgefüllten Pfeilende dargestellt. Durch einen Nachrichtenfluss werden immer zwei Teilnehmer des Geschäftsprozesses verbunden. Dargestellt werden diese durch unterschiedliche Pools. Die Verknüpfung geschieht entweder direkt oder durch die Verknüpfung mit einem Flow Object innerhalb eines Pools. Mit Hilfe einer Assoziation können mit einem Flow Object Artefakte assoziiert werden. Dargestellt wird eine Assoziation durch eine gepunktete Linie. Das Pfeilende einer Assoziation beschreibt die Flussrichtung [151].

2.7.7 Artefakte

Artefakte ermöglichen es Modellierern, weitere Informationen in einem Geschäftsprozess zu beschreiben. Die Struktur des Geschäftsprozesses wird dabei nicht verändert. BPMN spezifiziert zwei Typen von Artefakten, stellt es dem Modellierer aber frei, weitere Typen selbst zu erstellen. Dabei stellt eine Gruppe eine Gruppierung von Elementen dar. Sie wird durch ein gestricheltes, abgerundetes Rechteck repräsentiert. Eine Gruppierung ist lediglich ein grafisches Element und beeinflusst den Ablauf des Prozesses nicht. Sie wird zur Dokumentation bzw. zur Veranschaulichung von Zusammenhängen zwischen einzelnen Elementen im Geschäftsprozess verwendet. „Text-Annotations“ bieten dem Modellierer die Möglichkeit, zusätzliche Informationen für den Betrachter eines Diagramms bereitzustellen. Sie können durch eine Assoziation mit einem beliebigen Element verbunden werden. Auch können externe Dokumente mit den Artefakten in den Diagrammen verlinkt werden [152].

2.7.8 Zusammenfassung Basis-BPMN-Elemente

Die Grundelemente von BPMN entsprechen weitgehend den Flussdiagrammen. Diese bestehen aus Start- und Endereignissen sowie aus Aktivitäten, die durch Sequenzflüsse miteinander verbunden sind. Dazwischen sorgen Gateways dafür, dass Verzweigungen und Ausnahmefälle modelliert werden können. Die Darstellung verschiedener, am Prozess beteiligter Personen oder Systeme wird durch die Anordnung in Bahnen ermöglicht. BPMN umfasst mehr als 100 Elemente [152]. Viele Experten [152], [153] haben bereits zu den Elementen Diskussionen geführt und sind zu dem Ergebnis gekommen, dass die Notation sinnvoll eingegrenzt werden muss [105], [151]. So kommt es in diesem Fall nicht auf die Geschäftsprozessmodellierung an, sondern auf die Modellierung von Testprozessen. So müssen nicht nur BPMN-Elemente eingegrenzt werden, sondern BPMN muss um testspezifische Stereotypen erweitert werden (siehe Kapitel 3). Auch sollen die zu ziehenden Eingrenzungen dazu beitragen, Anwendern (Fachbereichen) das Modellieren von Testprozessen so einfach wie möglich zu machen. Sie müssen in der Lage sein, ohne langwierige Methoden- und Prozessschulungen BPMN-Modelle und Testmodelle zu verstehen und ggf. auch selbst anzufertigen und anzupassen. Auch sollen sie in die Lage versetzt werden, Testfälle selbstständig auszuführen. Das bedeutet, dass Fachbereiche aktiv an der Modellierung von Testfällen mitwirken müssen. Durch die Ausdrucksstärke von BPMN und durch den

breiten Sprachumfang ist die Modellierung von fachlichen Prozessen in leicht verständlicher Form möglich. Zugleich ist sie hinreichend präzise und erweiterbar für eine technische Umsetzung der Prozesse in der Softwareentwicklung.

Neben der Erweiterung des Metamodels muss gleichzeitig die Spezifikation von BPMN so erweitert werden, dass auch ein Testfallgenerator die modellierten Elemente interpretieren kann.

2.8 BPMN-Diagramme

Neben dem Prozessdiagramm und dem Kollaborationsdiagramm definiert BPMN 2.0 zwei weitere Diagrammtypen (Choreographiediagramm, Konversationsdiagramm), welche an dieser Stelle vollständigkeitshalber erwähnt werden [105], [162]. Beide dienen der Veranschaulichung der Interaktion zwischen verschiedenen Prozessteilnehmern. Der Schwerpunkt dieser Dissertation fokussiert die Prozess- und die Kollaborationsdiagramme.

2.8.1 Prozessdiagramm

Ein Prozessdiagramm stellt den Ablauf von Arbeitsschritten innerhalb einer Organisation dar. Die Beschreibung kann so detailliert und präzise sein, dass sie mit Hilfe einer Prozessausführungseinheit, wenn ein Modellierungswerkzeug eine solche zur Verfügung stellt, ausgeführt werden kann. Das Diagramm enthält dabei lediglich den Sequenzfluss sowie Datenobjekte und Artefakte, jedoch keine Nachrichtenflüsse. Diese werden im Kollaborationsdiagramm dargestellt. Das Diagramm enthält lediglich einen Pool, der jedoch in mehrere Lanes eingeteilt werden kann. Im Vergleich zu den vorigen Versionen von BPMN haben sich die grundlegenden Elemente für die Modellierung eines Prozesses nicht geändert [152]. Testprozesse können, genauso wie Geschäftsprozesse, mittels BPMN modelliert werden, da sie das erwartete Verhalten eines Softwaresystems wiedergeben.

2.8.2 Kollaborationsdiagramm

Eine Kollaboration stellt das Zusammenspiel mehrerer Partner (Softwaresysteme oder auch Personen) mittels Nachrichtenaustausch dar. Die beteiligten Softwaresysteme werden durch Pools modelliert. Die Pools können als sogenannte Blackbox-Pools gekennzeichnet werden, ohne den zugehörigen Prozess zu modellieren. Die Nachrichtenflüsse beginnen dann am Rand des Pools.

Die Pools können aber auch als Whitebox-Pools modelliert werden. In diesem Fall wird der Prozess in den Pool eingezeichnet (mit Hilfe eines Prozessdiagramms). Es kann eine weitere Unterscheidung in öffentliche Prozesse und private Prozesse vorgenommen werden. In öffentlichen Prozessen werden nur die Elemente modelliert, die am Nachrichtenaustausch mit Softwaresystemen beteiligt sind. In privaten Prozessen wird der komplette Prozess ausmodelliert.

2.8.3 Fachliche Modellierung von Testprozessen und die Vorgehensweise im Fallbeispiel dieser Dissertation

Die Vorgehensweise und die Zusammenarbeit zwischen den Bereichen bei der Spezifikation von Testfällen und bei der anschließenden Testfallmodellierung lassen sich am einfachsten anhand einer Grafik erläutern.

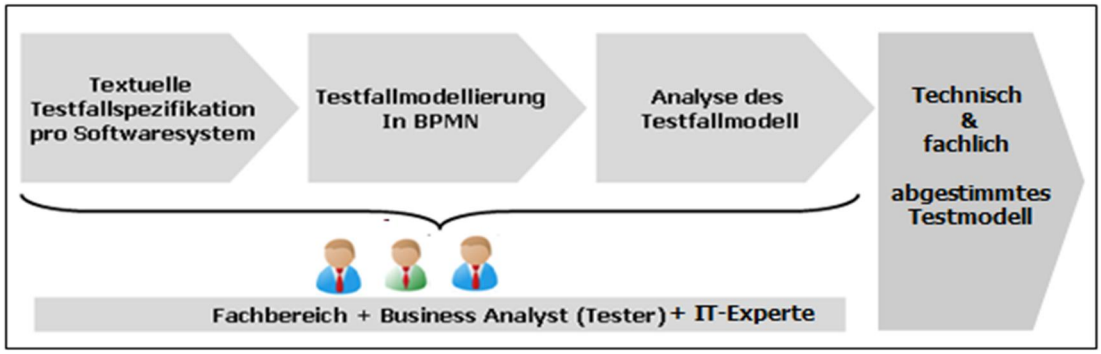


Abbildung 15: „Best Practice"-Vorgehensweise bei der Testfallspezifizierung bis zum Testmodell

Sowohl an der Testfallspezifikation (textuelle Testfallbeschreibung) als auch an der Analyse der Ergebnisse der Testfallspezifikation arbeiten einerseits Personen aus dem Fachbereich und andererseits Business-Analysten und IT-Experten. Das ist ein in der Praxis bewährtes Vorgehen, deshalb kann es mit zur „Best Practice" gezählt werden, so ist sichergestellt, dass sowohl Fachbereiche als auch IT-Experten und Business-Analysten schon zu Beginn in die Modellierung involviert werden und dadurch frühzeitig Missverständnisse und Fehler im Testmodell gemeinsam identifiziert werden können.

Die Geschäftsprozesse eines Unternehmens befinden sich oft nur in den Köpfen der Mitarbeiter. Die Prozesse werden durch die Mitarbeiter gelebt, sind jedoch nicht dokumentiert. Jeder Mitarbeiter kennt einen Ausschnitt des kompletten Prozesses und zwar genau den Teil, in den er involviert ist. Ein applikationsübergreifender End-to-End-Testprozess kann deshalb nur durch die Befragung vieler Mitarbeiter der verschiedenen Fachbereiche ermittelt werden. Dementsprechend müssen auch alle beteiligten Fachbereiche in die Modellierung dieser Testprozesse involviert werden. Da die beteiligten Fachbereiche allerdings keine Kenntnisse [105] in der Geschäftsprozessmodellierung haben, existiert meistens der Business-Analyst, der als Bindeglied zwischen Fachbereichen und IT agiert, um die applikationsübergreifende Modellierung von Geschäftsprozessen zu übernehmen. Die so entstehenden Testmodelle haben gleichzeitig den Vorteil, dass sie das Verhalten des Softwaresystems und auch der agierenden Prozesse transparent für alle beteiligten Parteien darstellen.

Im Rahmen dieser Dissertation und zum besseren Verständnis werden einmalig textuelle Beschreibungen zu den Testspezifikationen angefertigt, welche ein Business-Analyst als Basis seiner Modellierung nutzen kann. An dieser Stelle muss der Fachbereich (bei ihm liegt das Wissen über den detaillierten Prozessablauf) einen Business-Analysten in die Spezifikation der Testfälle involvieren, so entsteht Schritt für Schritt ein System- und Prozessverständnis sowohl beim Fachbereich als auch beim Business-Analysten. Anschließend werden die Fachbereiche, während der Business-Analyst die Prozesse modelliert, mit der BPMN-Notation vertraut gemacht. Fachbereiche sollen so einen Einblick in die Geschäftsprozessmodellierung bekommen und gleichzeitig auf Fehler und Widersprüche di-

rekt hinweisen. Ziel ist es, die Modelle so einfach wie möglich, aber trotzdem so präzise wie nötig zu halten. Die textuelle Testfallbeschreibung dient im Rahmen dieser Dissertation nur zum Verständnis, sie entfällt, wenn Testfallbeschreibungen direkt in Testmodellen erfasst werden und daraus dann direkt Testfälle generiert werden können, was ja auch das Ziel von MBT-Testmethoden ist.

2.9 Zusammenfassung – Kapitel Stand der Technik

Das Kapitel fasste die aktuellen Forschungsfragen zusammen und gab einen Überblick über die bisherigen, aus der Forschung und der Praxis bekannten Ansätze. An dieser Stelle muss eine zwingende Unterscheidung vorgenommen werden, was schon vorhanden ist und was diese Dissertation lösen muss:

a) Die bisherigen Testmethoden und -werkzeuge sind zumeist in der Domäne der Embedded Systems erprobt und auch evaluiert, [12] und [13] stellten in zwei Publikationen einen theoretischen Ansatz auch für ERP-Systeme vor. Beide dort vorgestellten Ansätze sind nicht erprobt und basieren auf theoretischen Gegebenheiten [30].
b) In den meisten Fällen haben die hier betrachteten Testwerkzeuge ihren Schwerpunkt in der Modellierung von Testfällen auf Modultestebene (Komponentenebene). Diese erfordern ein hohes Maß an programmiertechnischem Know-how.
c) Der Eindruck des Autors dieser Dissertation ist, dass die Werkzeuge von Ingenieuren für Ingenieure konzipiert sind.
d) Modellierungsnotationen sind sehr umfangreich und komplex, von UML über QML bis hin zu Markov-Ketten. Diese sind zwar für Embedded Systems ideal, allerdings für andere Softwaredomänen nur sehr schwierig zu vermitteln, da hier eine andere Zielgruppe (Fachbereiche mit fachlichem Prozess-Hintergrund) angesprochen werden muss als technische Softwareingenieure.

Was jetzt fehlt und durch diese Dissertation gelöst werden muss:

a) Was in den bisher analysierten Betrachtungen nicht ausreichend beleuchtet wurde, ist eine Testmethode, die sich auf das Testen auf Anwenderebene und auf Geschäftsprozesse (End-to-End) konzentriert.
b) Das Testen soll nicht mehr nur Embedded Systems fokussieren, sondern auch betriebswirtschaftliche Anwendungssysteme (ERP-Systeme), die geschäftsprozessgetrieben sind.
c) Dadurch verändert sich gleichzeitig auch der Blickwinkel des Testers. Nicht nur einzelne Komponenten sollen getestet werden, sondern die Applikation im Verbund mit allen anderen angebundenen Applikationen. Es geht um das Testen verteilter Applikationen.
d) Das erfordert Testfälle, die weitreichender formuliert sein müssen, daher müssen an dieser Stelle bei der Testfallerstellung Fachbereiche miteinbezogen werden. Genauer: Fachbereiche müssen Testfälle so spezifizieren, dass auch Schnittstellendaten einbezogen werden. Sprich: Was für eine Datei wird an Schnittstelle X erwartet und mit welchem Inhalt und wie ist diese evtl. noch für den Test zu manipulieren?

e) Dies erfordert allerdings eine für den Fachbereich auch einfach zu verstehende Modellierungsnotation, wenn es dann um die Modellierung von Testfällen geht.
f) Testfalldefinition auf BPMN-Ebene: Die Testfalldefinition erfolgt nicht durch Ingenieure, die technisch versiert sind, sondern auch durch den Fachbereich mit einem Blick auf die Prozesssicht. Vorhandene Notationen setzen einen technisch versierten Experten voraus, da sie sich mehr auf Steuerlogiken und Algorithmen beziehen, BPMN hingegen nicht.
g) Das Ziel ist es, einen Geschäftsprozess einheitlich über mehrere Applikationsinstanzen hinweg integrativ modellbasiert zu testen.
h) Bisherige BPMN-Notationen sind ideal für die Geschäftsprozessmodellierung. Allerdings ist diese für die Testmodellierung nicht ausreichend spezifiziert. Dadurch ist es vonnöten, die Spezifikation von BPMN um zusätzliche Testspezifikationen zu erweitern.

2.10 Probleme klassischer Testautomatisierungsansätze

Marc Utting und Bruno Legeard haben sich intensiv mit der Thematik des Softwaretestens beschäftigt [1]. Die Autoren beschreiben das Testen als eine Aktivität, bei der die Produktqualität evaluiert wird. In Bezug auf das Testen von Softwaresystemen bedeutet dies, dass das globale Ziel das Messen der Qualität des zu testenden Systems (SUT) ist. Die Anforderungsspezifikationen des SUT dienen dabei als Prüfreferenz, um die korrekte Umsetzung der Anforderungen durch das Softwaresystem bzw. Abweichungen zwischen spezifiziertem und tatsächlichem Verhalten des Softwaresystems identifizieren zu können. Die Herausforderung besteht darin, die Tests auszuwählen, die in einer Testphase und einer vorgegebenen Frist auszuführen sind. Allerdings lässt sich die Anzahl der auszuführenden Testfälle für eine vollständige Prozessabdeckung mit keinem vertretbaren Aufwand erreichen. Das Ausführen manueller Testfälle erfordert viel Zeit. Die gesamte Testdauer ist dabei von der Anzahl und der Güte der Testläufe abhängig. Demnach steigt der Testaufwand proportional zum Umfang des Systems [1]. Dies führt dazu, dass das manuelle Testen bei umfangreicheren Systemen schnell an seine Grenzen stößt. Was in der Praxis auch zu oft unterschätzt wird, ist die Tatsache, dass das Wiederholen von Testsitzungen für Tester sehr ermüdend sein kann, wodurch sich unbewusst Fehler während der Testausführung einschleichen können [77]. Auch das Reproduzieren des jeweils exakt gleichen Testvorgangs bspw. durch die Einhaltung der gleichen Testgeschwindigkeit und durch die aufwendige Vorbereitung der Testumgebung, kann schwierig umzusetzen sein [66]. Das risikobasierte Testen kann in der Praxis auch durch Testautomatisierung unterstützt werden. Dabei wird in der Praxis zwischen dem Capture & Replay und der deskriptiven Programmierung unterschieden. Eine weitere Methode ist das Capture & Replay der GUI, welches nur die graphische Benutzeroberfläche aufzeichnet. Verändern sich die Bedienelemente der graphischen Oberfläche, so können die Positionen nicht mehr identifiziert werden. Bei häufigen Änderungen an der Benutzerschnittstelle kann die durch die Automatisierung gewonnene Zeitersparnis so an Nutzen verlieren. Das Capture & Replay ist eine sehr schlechte Variante der Testautomation [66], [77].

Beim skriptbasierten Testen erweist sich der hohe Aufwand, welcher für die Implementierung der Skripte notwendig ist, als nachteilig, aber gleichzeitig sind diese

Testskripte robuster gegenüber Änderungen am zu testenden Softwaresystem. Jeder einzelne Schritt muss im Skript implementiert sein, das verlangt von einem Tester sehr gute Programmierfähigkeiten. Auch erfordert die Testskriptprogrammierung eine hohe fachliche Expertise bezogen auf die zu testenden Software (mit Testmodellen würde hier eine Vereinfachung eintreten) [3]. Die fachliche Expertise kann jetzt eingesetzt werden, um wiederverwendbare Bibliotheken mit Hilfe der Skripttechniken zu erstellen. Beim schlüsselgetriebenen Testen werden Funktionsbibliotheken erstellt, in denen Testsequenzen als Komponenten programmiert sind. Diese bilden kleine Funktionsbausteine, die aus verschiedenen Stellen im Programm aufgerufen werden können. Die Umsetzung einer guten Testbibliothek kann einen erheblichen Aufwand erfordern [3], so dass der Einstieg in diesen Testansatz sehr sorgfältig und bspw. gemeinsam mit Testmodellen und Generatoren dieser Modelle umzusetzen ist.

2.11 Präzisierte Aufgabenstellung – Grenzen klassischer Tests

Wie sieht nun die konkretisierte Aufgabenstellung aus? Das letzte Kapitel hat einen Überblick über das Thema Testen und über die verschiedenen Testvorgehensweisen und Methoden gegeben. Mit dem Wissen aus diesem Kapitel kann jetzt die Aufgabenstellung präzisiert werden.

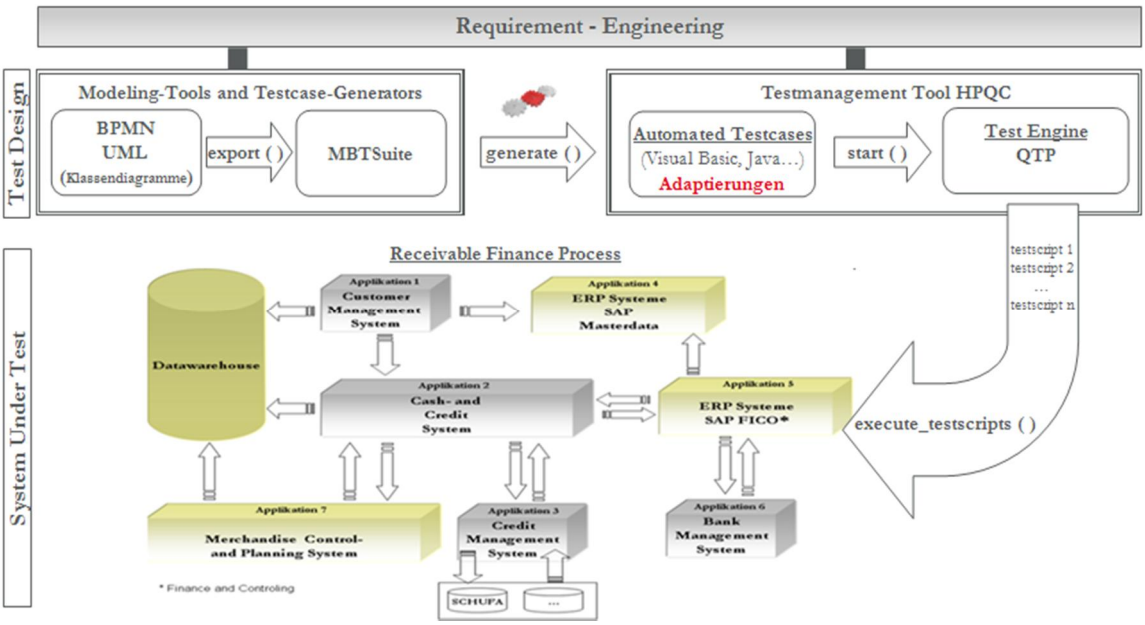


Abbildung 16: Präzisierte Aufgabenstellung: Lösung und beteiligte Test- und Modellierungswerkzeuge

Folgende Problembeschreibung aus der Praxis soll verdeutlichen, wie schwierig es ist, mit klassischen Testmethoden in verteilten Applikationen eine hohe Testabdeckung zu ermöglichen.

Das Beispiel, das durchgängig in dieser Dissertation genutzt werden soll, ist der B2B-Debitoren-Prozess (IT-gestützter Zahlungsabwicklungsprozess), welcher auf den ersten Blick ein scheinbar einfacher Prozess ist. An diesem Prozess sind mehrere Applikationen beteiligt. Funktioniert hier eine Applikation allerdings nicht, wie die Spezifikation und der Geschäftsprozess es vorgeben, kann das zu schwerwiegenden Fehlern führen.

Bisher werden diese verteilten Applikationen mit klassischen Testprozessen (Modultest, Integrationstest, Regressionstest und Abnahmetest) und Methoden getestet [10]. Allerdings werden nur selektiv Testfälle ausgewählt und durchgeführt, es handelt sich also um ein risikobasiertes Testen (Kapitel 2.2.4). Der Grund hierfür sind die sehr ausgeprägten Geschäftsprozesse in verteilten BIS. Jeder dieser Geschäftsprozessschritte erfordert mindestens einen Testfall, der diesen Geschäftsprozess natürlich so authentisch wie möglich nachbildet.

Das risikobasierte Testen birgt allerdings Gefahren, denn wichtige Pfade können beim risikobasierten Test übersehen oder gar vergessen werden. Eine Herausforderung besteht auch bei der Priorisierung der Testfälle, hier besteht die Gefahr, dass bei der Auswahl wichtige Testfälle durch Versehen oder Fehler mit einer niedrigen Priorität versehen werden. So kann es passieren, dass kritische Testfälle während des Testens aus zeitlichen Gründen nicht ausgeführt werden. Ein weiterer Nachteil risikobasierter Tests ist die nur vage Schätzung von Testabdeckungsgraden.

Die Zielsetzung dieser Dissertation fasst die obige Grafik visuell zusammen. Existierende BPMN-Modellierungswerkzeuge müssen jetzt dahingehend untersucht werden, inwieweit sie sich auch zur Testfallmodellierung eignen. Dabei steht das gewählte Beispiel exemplarisch für betriebliche Informationssysteme.

Fragen, die beantwortet werden müssen, sind: Wie werden die Schnittstellen in BPMN dargestellt? Wie wird der Datentransfer gekennzeichnet? Ein Testfall muss auch prüfen, ob die korrekten Testdaten übergeben werden: Wie wird dies in BPMN spezifiziert? Es müssen alle modellierten geschäftskritischen Pfade durchlaufen werden, um sicherzustellen, dass die Testabdeckung vollständig ist. 100% ist eine fiktive Zahl, da Fehler sich trotzdem in einer Applikation befinden können, allein schon durch falsch formulierte Anforderungen [123].

Sind die ersten Testfälle generiert, muss in einem weiteren Schritt untersucht werden, inwiefern die generierten Testfälle sich direkt in ausführbare Testskripte überführen lassen. Weiterhin soll untersucht werden, ob bisherige Testskriptbibliotheken sich weiterhin nutzen lassen und wie ein Transfer der generierten Testskripte ins Testmanagementtool erfolgen kann. Was aus vorhandenen Werkzeugen übernommen werden kann, ist bspw. die Modellierungskomponente für Aktivitätsdiagramme und BPMN. Im Speziellen bietet sich die Nutzung der folgenden MBT-Werkzeuge an: IBA und MBTSuite; als Testmanagementtools werden HPQC sowie HP QTP für die Programmierung der Testskripte genutzt. Auch muss überprüft werden, ob die Schnittstellen mit den HP-Testwerkzeugen angepasst werden müssen. Die notwendigen Erweiterungen der Testtools ergeben sich aus dem Kapitel der technischen Realisierung und der Evaluierung.

Anhand der Kombination dieser Werkzeuge soll untersucht und auch unter Praxisbedingungen erprobt werden, ob diese Methode die Anforderungen der Dissertation erfüllt. Die Erprobung erfolgt anhand des Fallbeispiels, welches im ersten Schritt nur den IT-gestützten Zahlungsabwicklungsprozess beinhaltet. Natürlich muss es möglich sein, dann jeden beliebigen Geschäftsprozess mit dieser Methode zu testen.

Präzisierte Aufgabenstellung

Eine präzisierte Anforderungsliste ist schon Bestandteil von Kapitel 1.4 „Anforderungen an diese Dissertation“. Jede dieser Anforderungen wird in Kapitel 5 implizit durch die Evaluierung geprüft.

Zusammenfassend lassen sich die Anforderungen an diese Dissertation so formulieren: Mit BPMN in verteilten BIS MBT realisieren. Genauer: Die Testmethode soll nicht nur einen speziellen Prozess abdecken, sondern sie richtet sich an alle Geschäftsprozesse in verteilten BIS. Mit einer modellbasierten Testmethode sind verteilte betriebliche Informationssysteme im Verbund mit angebundenen Schnittstellen zu testen.

Klassische Testprozesse reichen nicht aus, um diese Prozesse in allen Varianten in einer vertretbaren Zeit zu testen. Testautomatisierung a la Capture & Replay schafft kurzfristig Abhilfe bei Routinetätigkeiten. Allerdings bergen die ständigen Releasewechsel applikationsübergreifend die Gefahr, dass diese Testskripte immer neu angepasst werden müssen. Dies generiert Aufwand. Eine Alternative bietet die deskriptive Skripttechnologie, und zwar gepaart mit Testmodellen.

Auch verfolgt diese Dissertation nicht das Ziel einer Big-Bang-Einführung und einer Etablierung der MBT-Testtechnologie im Bereich der verteilten BIS. Sondern es wird die Strategie der kleinen Schritte gewählt. Erst einmal erfolgen die Einführung und die Nutzung des MBT in einem kleinen Team, das zunächst aus Experten besteht (Testexperte, Fachbereich, evtl. Programmierer). Es sollte in einem begrenzten Umfeld (IT-gestützter Zahlungsabwicklungsprozesses im B2B (Debitoren-Prozess)) eingesetzt werden, gleichzeitig soll die Methode fortlaufend auf Praxistauglichkeit evaluiert werden. Genau dieser Ansatz soll im Rahmen dieser Dissertation verfolgt werden.

3 Eigener Lösungsansatz

Dieses Kapitel beschreibt den eigenen Lösungsansatz, dabei werden die Themenschwerpunkte in insgesamt vier Teilen erläutert. Im ersten und zweiten Teil sind die für die Dissertation relevanten Geschäftsprozesse aus dem Tagesbetrieb von B2B-Geschäftsprozessen Gegenstand der Erläuterung. Die Problembeschreibung skizziert abstrakt einen Beispielprozess, welcher verdeutlicht, mit welchen Herausforderungen Testprozesse sich in den verschiedensten Unternehmen (B2B) auseinandersetzen müssen.

Als Demonstration und durchgehendes Fallbeispiel dieser Dissertation dient der betriebswirtschaftliche Debitoren-Prozess bezogen auf B2B. Debitoren sind Forderungen gegenüber Kunden (B2B-Unternehmen), dabei steht ein Debitoren-Prozess stellvertretend für mehrere hintereinander ablaufende und voneinander abhängige, auszuführende Geschäftsprozesse im Unternehmen, welches Produkte/Waren verkauft. Ziel dieses Geschäftsprozesses ist die vollständige IT-gestützte Zahlungsabwicklung in B2B-Unternehmen [81], [91].

Anschließend wird ein Lösungsansatz vorgestellt, welcher unabhängig von der eingesetzten Softwarelösung und Systeminfrastruktur das Testen von End-to-End-Geschäftsprozessen mit Hilfe von modellbasierten Testmethoden ermöglicht. Der IT-gestützte Zahlungsprozess dient hier als durchgehendes Fallbeispiel, da dieser viele Facetten von Geschäftsprozessen abdeckt, welche gleichzeitig wiederum in verschiedenen Prozesspfaden münden und das Testen solcher Prozesse sehr aufwendig machen.

Der hier zu beschreibende Lösungsansatz zum Testen ist ein „Best Practice“-Ansatz, welcher dann am Fallbeispiel des IT-gestützten Zahlungsprozesses im zweiten Teil dieses Kapitels technisch umgesetzt und demonstriert werden soll [33]. Dieser Ansatz beinhaltet Wissen und Erfahrungen sowohl aus dem Blickwinkel der IT (Entwicklung und Test) als auch aus der Prozesssicht der jeweiligen Mitarbeiter aus verschiedenen Fachbereichen.

Da dieser „Best Practice“-Ansatz ein generelles Testvorgehen (regressiv) bei End-to-End-Geschäftsprozessen beschreibt, kann er als Referenz und Empfehlung weiteren Unternehmen mit B2B-Geschäftsprozessen zur Verfügung gestellt werden. End-to-End-Testen bedeutet, dass ein Prozess von seiner Entstehung bis zu seinem Abschluss applikationsübergreifend getestet wird. Der regressive End-to-End-Test sorgt dafür, dass bei jeder Softwareänderung, bei jeder Softwareanpassung und bei jedem Softwareupgrade eines am Prozess beteiligten Softwaresystems sichergestellt ist, dass die Softwaresysteme immer noch so funktionieren, wie in der Spezifikation vorgesehen. Somit kommt dem Regressionstest eine besondere Bedeutung zu.

Zum erfolgreichen Testen von End-to-End-Geschäftsprozessen gehören auch die notwendigen Systemlandschaften, welche auch Gegenstand dieses Kapitels sein werden. Ein „Best Practice“ auch bei den Systemlandschaften soll demonstrieren, welche Systemlandschaften sich in der Praxis bewährt haben. Der letzte Teil dieses Kapitels behandelt dann auch die konkret einzusetzenden Werkzeuge, welche für die Umsetzung und die Realisierung eines modellbasierten Testansatzes benötigt werden.

Neben den eingesetzten Softwarelösungen und Systeminfrastrukturen werden auch Grenzen und Probleme klassischer Testansätze bei vollautomatisierten End-to-End-Geschäftsprozessen erläutert.

Kapitel 4 durchleuchtet die an der Lösung beteiligten Testwerkzeuge und die Vorgehensweise bei der Umsetzung der Methode. Auch werden die notwendigen Voraussetzungen demonstriert.

3.1 IT-gestützter Zahlungsabwicklungsprozess und vorgeschlagene Applikationen

Die Art, auf die ein Unternehmen geschäftliche Beziehungen unterhält, ist je nach Unternehmen und Branche unterschiedlich. Dabei werden zwei Arten von Kunden unterschieden, B2B (Business-to-Business) und B2C (Business-to-Consumer). B2B-Unternehmen verkaufen ihre Produkte und Dienstleistungen direkt an andere Unternehmen, wie bspw. in der Medizinbranche, in der Pharma-, in der Automobilindustrie oder im Handel. Diese wiederum verkaufen ihre Produkte an andere produzierende Unternehmen und nicht direkt an Endkunden. Ein Unterschied zwischen den B2B- und den B2C-Kunden ist, dass B2B-Unternehmen i.d.R. eine Einkaufsabteilung mit professionellen Einkäufern haben, deren einzige Aufgabe darin besteht, einzukaufen – Woche für Woche, Tag für Tag, von morgens bis abends. Verkaufsabteilungen mit Kundenmanagern wiederum versuchen, die „Einkäufer“ zu betreuen und diese an die Unternehmen zu binden. So erzielten B2B-Unternehmen im Jahr 2012 allein in Deutschland einen Gesamtumsatz von 870 Mrd. Euro[42]. Dadurch wird deutlich, welche Wichtigkeit funktionierende Geschäftsprozesse in Unternehmen haben, zumal die Zahlungsabwicklung größtenteils IT-gestützt erfolgt.

Der Debitoren-Prozess ist demnach als ein abstrakter Begriff zu verstehen, welcher aus mehreren Prozessen und Unterprozessen besteht. Dabei stehen B2B-Unternehmen ständig vor großen Herausforderungen bei der Verwaltung und der Pflege ihrer Debitoren, da die Anzahl oft in die Hunderttausende[43] gehen kann. Unabhängig davon, um welche Branche es sich handelt (Handel, Telekommunikation, Versandhäuser usw.), muss derjenige, der für die Debitorenbuchhaltung (Zahlungsabwicklung) verantwortlich ist, zentrale Aufgaben wahrnehmen, wie die Geringhaltung der Debitorenaußenstände, die Pflege eines effektiven Mahnwesens und die Vermeidung von Verlusten aus zweifelhaften Forderungen. Dafür setzen Unternehmen heutzutage vollautomatisierte Geschäftsprozesse ein, welche diese Geschäftsprozessketten bis zum Zahlungsabwicklungsprozess unterstützen.

IT-gestützte Zahlungsabwicklungsprozesse in B2B-Unternehmen ähneln sich branchenübergreifend und unterscheiden sich nur minimal in der einzusetzenden Systemlandschaft und Softwarelösung. Viele Unternehmen nutzen heute SAP (Software – Produkte – Anwendungen), eines der weltweit führenden[44] Enterprise-Resource-Planning-(ERP)-Systeme, zur Verwaltung ihrer Geschäftsprozesse. Eine externe Verbindung zu Handelspartnern und Kunden ist für ein effizientes Funktionieren der SAP-Systeme unerlässlich, und dies lässt

[42] Quelle: IFH RETAIL CONSULTANTS, Umsätze ohne Mehrwertsteuer.
[43] Anzahl der Kunden bei Handelskonzernen (Quelle: EHI Handelsdaten).
[44] www.sap.com.

sich leicht durch die nahtlose Integration in Systeme wie Kundenmanagement- oder Kreditmanagementsysteme, welche auf verschiedenen Plattformen basieren, erreichen.

Einen ersten Einstieg in das Thema Debitoren-Prozesse und IT-gestützte Zahlungsabwicklung im B2B bietet Abbildung 17. Diese illustriert gleichzeitig eine „Best Practice"-Systeminfrastruktur, welche sich bei vielen B2B-Unternehmen in der Praxiswelt bewährt hat [33]. Zusätzlich demonstriert sie, wie auf Softwaresystemebene unterschiedliche Softwarelösungen miteinander agieren, um Geschäftsprozesse zu gestalten und um Wertschöpfung zu schaffen. B2B-Unternehmen nutzen für die Abwicklung ihrer Kernprozesse nicht unbedingt die gleichen Softwarelösungen, da sich auch die Unternehmen an einem „Best Practice" in der jeweiligen Branche orientieren. So können Unternehmen zwar miteinander im Wettbewerb stehen, nutzen aber für die Abwicklung ihrer Prozesse die gleichen Softwarelösungen. So bietet SAP genügend Optionen für viele Geschäftsprozesse von Unternehmen-Referenzmodellen gleicher oder unterschiedlicher Branchen.[45]

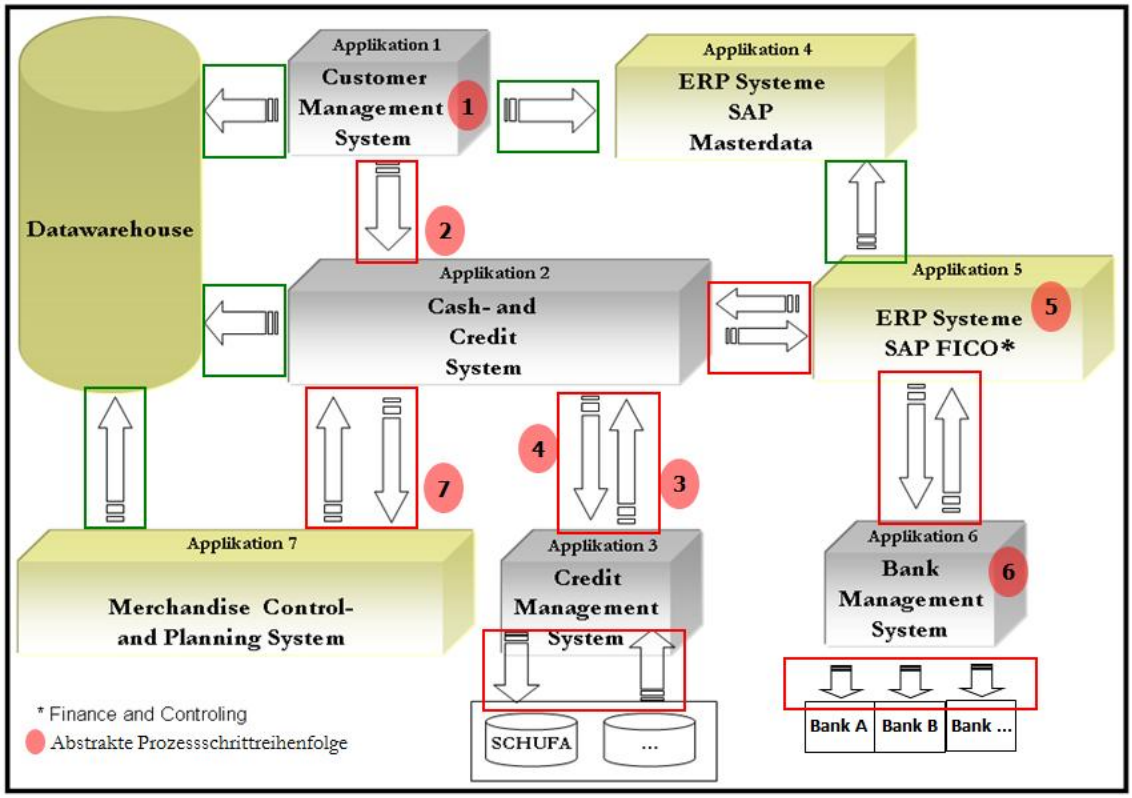


Abbildung 17: IT-gestützter Zahlungsabwicklungsprozess und die beteiligten Softwaresysteme [33], modifiziert durch den Autor dieser Dissertation

Die Abbildung bietet einen Überblick über die Softwaresysteme, welche für die Bearbeitung betriebswirtschaftlicher Prozesse eingesetzt werden [106]. Hinter diesen Applikationen stehen Softwarelösungen verschiedener Hersteller und Anbieter. Wobei die rot umrandeten Kästchen kritische Schnittstellen zeigen und die grün umrandeten die eher unkritischen Schnittstellen. Kritisch bedeutet in diesem Zusammenhang, dass geschäftskritische Daten zwischen den Softwaresystemen ausgetauscht werden. Geschäftskritische Daten und Prozesse sorgen für die Wertschöpfung in einem Unternehmen und ein Nichtfunktionieren

[45] Becker, J. (1996): Eine Architektur für Handelsinformationssysteme. In: Becker, J. et al. (Hrsg.): Arbeitsberichte des Instituts für Wirtschaftsinformatik, Nr. 46, Münster : Westfälische Wilhelms-Universität. https://www.wi.uni-muenster.de/de/forschung/arbeitsberichte.

dieser Schnittstellen bedeutet gleichzeitig, dass für ein Unternehmen dadurch ein Schaden entstehen kann.

Applikation 1 stellt bspw. ein Kundenmanagementsystem dar, welches von Siebel und Oracle auf der Basis von Java entwickelt und vertrieben wird. Siebel ist Marktführer[46] und bietet Kundenmanagementlösungen für Branchen wie Energie, Pharma, Automobilindustrie, Versicherungen und Handel. Viele B2B-Unternehmen, wie Metro, Tesco[47] oder auch Carrefour, setzen Siebel-Software ein, um die Verwaltung ihrer Kunden professionell zu gestalten. Mit diesem System können Unternehmen Kunden gewinnen, aber auch bestehende Kunden besser binden, mit dem Ziel, das Umsatzvolumen der Kunden zu steigern.

Applikation 2: Ein Kassensystem mit integriertem Kreditsystem. Kassensysteme haben bei B2B-Unternehmen zwei Funktionen. Sie sollen einerseits die verkauften Artikel erfassen, so dass ein Kunde die Möglichkeit hat, die eingekauften Produkte direkt zu bezahlen. Eine zweite Funktion ist die Lieferung von Kundenumsätzen sowohl an die Buchhaltungsprogramme wie SAP-FI als auch an das Kundenmanagementsystem. Kassensysteme werden speziell im Handel eingesetzt und andere Branchen ersetzen diese durch die im Standard von Kundenmanagementsystemen angebotenen Funktionalitäten zur Rechnungserstellung, welche es gleichzeitig erlauben, Rechnungen auch direkt per Schnittstelle elektronisch an den Kunden zu senden.

Applikation 3: Dieses Softwaresystem basiert in der Regel auf historischen Daten (Massendaten) und Einträgen aus externen, angebundenen Systemen. Die Berechnungen für ein Kreditlimit sind auf mathematische Verfahren und Regeln des Scoring zurückzuführen. Einflussfaktoren sind die externen Wirtschaftsauskunfteien, welche vielen Unternehmen als Basis für eine Bonitätsprüfung ihrer Geschäftspartner dienen. Kreditmanagementsysteme unterscheiden sich von Unternehmen zu Unternehmen in der jeweils konfigurierten Unternehmensrichtlinie[48] des Unternehmens. Jedes Unternehmen definiert Richtlinien für Geschäftsbeziehungen mit Geschäftspartnern. So können bspw. in einem Kreditsystem Regeln definiert werden, welche Bedingungen Geschäftspartner erfüllen müssen, um bei dem jeweiligen Unternehmen Waren/Artikel einkaufen zu können.

Applikation 4: Unternehmen, welche SAP für die Abwicklung von Geschäftsprozessen einsetzen, lagern Kundenstammsätze extern in einen zusätzlichen SAP-Mandanten aus. Der Grund hierfür ist einfach: SAP bietet Standardschnittstellen zu verschiedenen Softwaresystemen. Das hat den Vorteil, dass verschiedene Softwaresysteme in einer Unternehmenssystemlandschaft auf diese Kundenstammdaten zugreifen können; dazu benötigen sie nur eine Standardanbindung an einen SAP-Mandanten. Diese Vorgehensweise ist zwar nicht in allen Unternehmen Standard, gilt aber bei Unternehmen, die sie einsetzen, als bewährte Praxis.

Applikation 5: Dies ist ein Buchhaltungsprogramm, welches auf dem SAP-Modul „Finanzen“ (FI) basiert. Unabhängig davon, welches Softwaresystem konkret für die Buchhaltung eingesetzt wird, interagieren buchhalterische Geschäftsprozesse bei IT-gestützten Zahlungsabwicklungen gleich. Die Softwaresysteme unterscheiden sich hinsichtlich der aufzu-

[46] www.siebel.de
[47] Sowohl Tesco, Metro als auch Carrefour agieren auf dem B2B-Markt, wobei Carrefour mit über 450.000 Mitarbeitern einen der größten Konkurrenten der Metro Group (mit ca. 280.000 Mitarbeitern) darstellt.
[48] http://www.zis-online.com/dat/artikel/2008_9_261.pdf.

rufenden Transaktionen, allerdings nicht im auszuführenden Geschäftsprozess. In SAP unterscheiden sich die Geschäftsprozesse von denen des jeweiligen Unternehmens, welches den Prozess zur Zahlungsabwicklung abwickelt, nicht, da diese Standardprozesse der SAP sind. Diese Dissertation nutzt SAP-FI als Buchhaltungsprogramm, weil es der Standard für große B2B-Unternehmen ist und diese Unternehmen auf die Referenzprozesse von SAP zurückgreifen [142].

Applikation 6: Dies sind spezielle Programme, welche implizit aus Buchhaltungsprogrammen wie SAP-FI gestartet werden können. Die zentrale Aufgabe dieser Programme ist es, die Geldtransferprozesse zwischen den Bankkonten der Kunden und der Unternehmen abzuwickeln.

Verteilte betriebswirtschaftliche Anwendungssysteme, welche die Softwaresysteme aus Abbildung 17 darstellen, haben den Vorteil, dass die Komplexität auf einzelne Softwareteile verteilt wird, dadurch wird Flexibilität bei der Verteilung von Aufgaben erreicht und gleichzeitig konzentriert sich jede Applikation auf ihre Kernfunktionalitäten. Zugleich zeigt die Abbildung einen „Best Practice“-Ansatz und wie eine Systemlandschaft im B2B-Unternehmen, angelehnt an „Best Practice“ in [160] und auf die Praxiswelt übertragen, aussehen kann. Die Autoren dieser als Buch gebundenen Veröffentlichung sind nicht nur im Bereich des Software-Engineering und der Beratung von verschiedenen Unternehmen und Branchen tätig, sondern sie sind auch ausgewiesene Experten in der IT-Qualitätssicherung.

Ein spezielles Softwaresystem stellt das Data-Warehouse (DWH) dar, welches in der Lage ist, Datensätze von verschiedenen externen Softwaresystemen und von verschiedenen Transaktionen redundant zu speichern. Das DWH gilt als Historien-Datenbank (bspw. Teradata[49]) und ermöglicht verschiedene Auswertungsfunktionalitäten, die B2B-Unternehmen einsetzen, um eine bessere Kundenbindung zu erreichen und frühzeitig für Kunden maßgeschneiderte Produkte anzubieten. Solche Technologien[50] setzen bspw. auch Amazon oder die Otto Group ein, um ihren Kunden maßgeschneiderte Artikelangebote anzubieten. B2B-Unternehmen setzen hier auf dieselbe Technologie, um das Kaufverhalten ihrer Kunden (Unternehmen) zu analysieren und die Produktpalette darauf auszurichten. Dazu bedient sich ein Kundenmanagementsystem der im DWH abgelegten Informationen.

Zusätzlich illustriert die Abbildung den Daten- und Informationsfluss zwischen den einzelnen Softwaresystemen. Die kritischen Geschäftsprozesse sind dabei in allen B2B-Unternehmen vergleichbar, so stellen die rot umrandeten Pfeile in der Abbildung die kritischen Schnittstellen dar. Die grün umrandeten Pfeile stellen dabei Schnittstellen dar, die keine kritischen Daten senden oder empfangen, zumindest keine, die sofort in einem Softwaresystem zur weiteren Verarbeitung benötigt werden. Somit ist die Priorisierung der Schnittstellen im Debitoren-Prozess festgelegt, was vor allem beim End-to-End-Testen notwendig wird.

[49] Teradata ist der weltweit führende Lösungsanbieter für integriertes Data Warehousing.
[50] Teradata Magazine Volume 13, Number 2.

3.1.1 Kriterien für Priorisierungen

Da Testmanagern in der Regel schon bei der Planung zu wenig Zeit und Ressourcen zur Verfügung stehen und es unmöglich ist, alle möglichen Testfälle in einem Softwaresystem zu testen, ist es wichtig, eine Priorisierung vorzunehmen. Die Zeitknappheit ist der späten Softwarelieferung aus der Entwicklung geschuldet. Hinzu kommt, dass trotz einer guten Planung äußere Umstände (z. B. verzögerte Auslieferung der zu testenden Software) dazu führen können, dass die vorhandenen Mittel reduziert werden. Hier kann es sehr hilfreich sein, wenn die höher priorisierten Testfälle zuerst und die unwichtigeren Testfälle zuletzt (ggf. gar nicht) getestet werden. Ein paar Kriterien, nach denen Priorisierungen vorgenommen werden können [7], [10], [30], sind folgende:

- Häufigkeit der Nutzung der Geschäftsprozesse – analog dazu müssen die dazu passenden Testfälle priorisiert werden.
- Fehlerrisiko im Geschäftsprozess sowie die Sichtbarkeit von Fehlern für den Endanwender, was zu einer geringeren Akzeptanz der Softwarelösung führen kann.
- Priorisierung nach den Anforderungen, bspw.: Welche Anforderung ist am wichtigsten für die Software und soll deswegen zuerst getestet werden?
- Risikobehaftete Programmteile und Schnittstellen, welche bspw. zu Programmabstürzen bei Fehlern führen könnten.

Zu guter Letzt: Fehler, die zu einem Projektrisiko führen könnten. Daher muss eine sinnvolle Auswahl an Tests getroffen werden, um möglichst frühzeitig so viele kritische[51] Fehler wie möglich zu finden. Ziel der Priorisierung ist es, eine Gleichverteilung von Tests zu vermeiden, da kritische Teile in einem Softwaresystem bevorzugt getestet werden müssen. Bspw. müssen die Teile in einem Softwaresystem getestet werden, die einen hohen „Schaden“ im Fehlerfall verursachen können, und gleichzeitig soll das Risiko minimiert werden, dass Fehler in die Produktivumgebungen gelangen. Durch die Historie entstand dazu auch der Begriff des „risikobasierten Test“, welcher heute in der Praxis breite Anwendung findet [39], [59]. Gleichzeitig werden Testwerkzeuge (klassische Testautomatisierung) eingesetzt, um das risikobasierte Testen zu unterstützen. Risikobasiertes Testen bedeutet aber auch, dass Fehler bewusst in Kauf genommen werden und mit Hilfe von Priorisierungen während des Testens das Risiko, das große kritische Fehler in den Produktivbetrieb gelangen, nur minimiert wird.

3.1.2 Herausforderungen im Test

Als Debitoren werden Vertragspartner (Unternehmen) verstanden, für die Leistungen erbracht werden (bspw. Waren übergeben) und gegenüber denen dann Forderungen aus Lieferungen und Leistungen bestehen [142]. Die Debitorenbuchhaltung befasst sich demnach mit der Erfassung und Verwaltung aller offenen Forderungen gegenüber einem Vertragspartner. Zur Verwaltung und Erfassung von Debitoren werden Softwaresysteme genutzt, im Fallbeispiel SAP-FI. Um die Debitorenbuchhaltung zu aktivieren, muss ein Buchhalter den Zahllauf in einer Buchhaltungssoftware, im Fallbeispiel SAP-FI, starten. Die folgende Abbildung lehnt sich an Abbildung 17 an und zeigt, welche Prozesse in den einzelnen

[51] Siehe auch Kapitel 2 Stand der Technik – Klassifikation von Fehlern.

Softwaresystemen abgearbeitet werden. Die Nummerierung der Applikationen erfolgt analog zu Abbildung 17, wobei das Data-Warehouse hier als reines Auswertungssystem wahrgenommen wird.

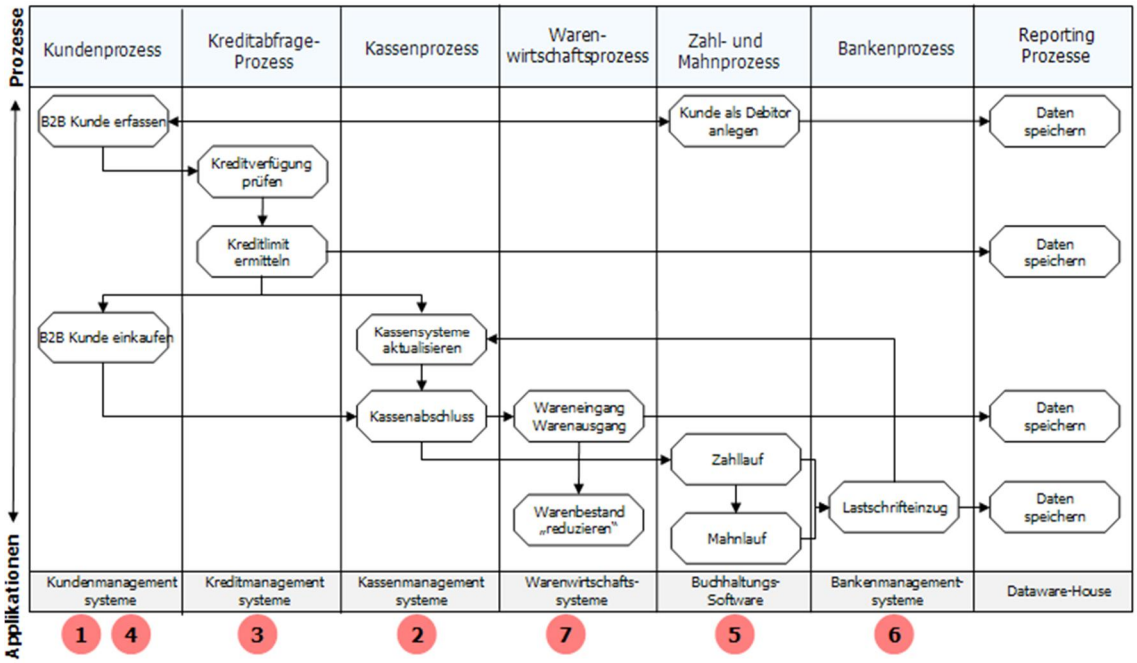


Abbildung 18: Agierende Prozesse im IT-gestützten Zahlungsabwicklungsprozess

Generell bestehen die Debitoren-Prozesse bei B2B-Unternehmen aus folgenden, aufeinander aufbauenden Geschäftsprozessen, wobei Abbildung 18 im Unterkapitel 3.1.3 etwas detaillierter erläutert wird.

- Kunden werden im Kundenmanagementsystem angelegt und werden in der Buchhaltung als Debitoren geführt. Kunden können Unternehmen aus verschiedenen Branchen sein. Bei bestehenden Kunden werden die Daten ständig mittels der angebundenen Systeme aktualisiert.
- Ein B2B-Unternehmen möchte wissen, mit welchen Unternehmen Geschäftsbeziehungen eingegangen werden sollen, deshalb werden Kunden im B2B einer Bonitätsprüfung unterzogen. Diese soll sicherstellen, dass das eigene Unternehmen keinen finanziellen Schaden erleidet, es soll aber gleichzeitig treuen und zuverlässigen Kunden ermöglichen, auf Rechnung und Kredit einzukaufen. Es kann sowohl auf eine SAP-basierte Kreditmanagementlösung als auch auf einen Drittanbieter zurückgegriffen werden. Im Rahmen dieser Dissertation wird ein externes Kreditmanagementsystem vorausgesetzt, welches sich vom SAP-Standard nicht unterscheidet. Es muss, ebenso wie eine SAP-basierte Lösung, bezogen auf die Unternehmenserfordernisse (Richtlinien) konfiguriert werden. Es ist ein wichtiges Kontrollinstrument bei Forderungen gegenüber Debitoren.
- Der Kreditabfrageprozess ist abhängig von den Kundenstammdaten und von der Anbindung an externe Wirtschaftsauskunfteien. Kreditabfrageprozesse sind sehr kompliziert und bedürfen bei einer Erläuterung einer eigenen Ausarbeitung. Der Autor dieser Dissertation beschränkt sich deshalb nur auf die Ergebnisse, welche dieses System liefert. So werden die ermittelten Kreditlimits sowohl an die Kundenmanagementsysteme als auch an die Kassensysteme übermittelt. Gleichzeitig

wird ein Kundenmanager, der dem jeweiligen Kunden zugeordnet ist, darüber informiert, ob und in welcher Höhe seinem Kunden ein Kredit gewährt worden ist. SAP-basierte Kreditmanagementlösungen agieren analog zur hier vorgestellten Drittanbieter-Lösung. Auch die SAP-basierte Lösung hat als Grundlage ihrer Entscheidungen die externen Wirtschaftsauskunfteien sowie die Unternehmensrichtlinien.

- Kundenmanager werden elektronisch über die Kreditlimits und die Entscheidung informiert. Bei Großkunden werden Kundenmanager eingesetzt, welche die Kommunikation mit dem Kunden übernehmen und sowohl Lieferbedingungen wie auch Konditionen aushandeln.
- Sobald Kunden Waren/Produkte eingekauft haben, werden sowohl die Umsätze (im Buchhaltungssystem jeweils dem Kunden über die Kundennummer zugeordnet) als auch der verkaufte Artikel (Warenwirtschaftssystem) in den jeweiligen Softwaresystemen verbucht. Der Umsatz mit den offenen Posten wird an die Buchhaltungssoftware (SAP-FI) transferiert. Die verkauften Produkte werden im Lagerbestand (Warenwirtschaftssystem) reduziert. Anschließend müssen die einem Kunden verkauften Produkte sowie die Umsatzhöhe dem DWH gemeldet werden.
- Kernprozesse der IT-gestützten Zahlungsabwicklung sind der Zahllauf- sowie der Mahnprozess, beide Kernprozesse werden explizit erläutert.
- Bankenprozesse werden durch den Zahllaufprozess gestartet und werten die im Zahllaufprozess ermittelten offenen Posten aus. Die Aufgabe dieses Prozesses ist es, die Geldbeträge aus den gemeldeten offenen Posten des Zahllaufs mittels elektronischen Lastschrifteinzugs durchzuführen und gleichzeitig die Kundensalden in SAP-FI auszugleichen. Sollte ein Lastschrifteinzug mangels Kontodeckung nicht ausgeglichen werden können, werden anschließen automatisierte Mahnprozesse angestoßen.

Diese abstrakte Kurzbeschreibung des Prozessablaufs verdeutlicht auf den ersten Blick den Umfang eines möglichen Geschäftsprozessablaufs und zeigt gleichzeitig den aufzubringenden Aufwand beim Testen (siehe ab Kapitel 3.5.), um analog dazu einen Test dieses Geschäftsprozessablaufs durchzuführen.

Auch die ständig wechselnden Geschäftsanforderungen im B2B der heutigen Zeit wirken sich direkt auf die jeweilige IT-Abteilung im Unternehmen aus. Unternehmen versuchen ständig, ihre Ausgaben[52] zu senken und gleichzeitig die Markteinführungszeiten ihrer Produkte zu verkürzen. Dadurch werden hoch integrative und heterogene Softwaresysteme (Oracle, SAP, Java usw.) eingesetzt, um den Tagesbetrieb im Unternehmen so IT-gestützt wie möglich zu unterstützen. Schnittstellen verschiedenster Softwaresysteme müssen miteinander kommunizieren und harmonieren.

Im Grunde ist es unwesentlich, welches Softwaresystem in der Systeminfrastruktur angepasst, erweitert oder verändert werden soll. Durch die Vernetzung und die Integration der Softwaresysteme muss immer sichergestellt werden, dass der Prozess dementsprechend auch als Einheit einwandfrei funktioniert. Und genau hier verbergen sich die grundlegendsten Probleme beim Testen. Bei jeder Änderung oder auch Anpassung in einem Software-

[52] Praxisleitfaden „Testen moderner SOA-Systeme“ von Gaurish Hattangadi (InfoSYS) – Gaurish Hattangadi agiert gleichzeitig seit August 2013 als Partner von vielen Unternehmen bei strategischen Themen.

system muss regressiv getestet und geprüft werden, ob die Geschäftsprozesse auch weiterhin einwandfrei funktionieren. Diese regressiven Tests erfordern einen sehr hohen Ressourcen- und Zeitaufwand, so dass mit heutigen klassischen Testansätzen nicht alle kritischen Schnittstellen und Prozesspfade mit einem vertretbaren Aufwand getestet werden können.[53] Mario Winter und Harry M. Sneed demonstrieren in [137] eindrucksvoll, wie klassische Testansätze bei verteilter und komplexer Software an ihre Grenzen (textuelle Erfassung von Testfällen, manuelle Testausführung) kommen. Im Folgenden werden die Softwaresysteme aus Abbildung 17 und die damit agierenden Prozesse aus Abbildung 18 zusammenhängend am Fallbeispiel erläutert, um den Testumfang zu verdeutlichen.

3.1.3 Prozessbeschreibung am abstrakten Fallbeispiel – Zuordnung des Prozesses zur Applikation

Abbildung 18 erläutert die einzelnen, mit Abbildung 17 korrespondierenden Geschäftsprozessabläufe. Wichtig an dieser Stelle ist nicht die konkrete Spezifikation des Prozessablaufs bspw. mit Testdaten, sondern an dieser Stelle genügt es zunächst, eine abstrakte Sichtweise auf die IT-gestützte Zahlungsabwicklung und die im Hintergrund agierenden betriebswirtschaftlichen Prozesse zu bieten. Die verschiedenen am Prozess beteiligten Softwaresysteme in Abbildung 18 sind zur besseren Unterscheidung analog zur Abbildung 17 durchnummeriert.

In Applikation 1 werden zunächst B2B-Kunden angelegt, gleichzeitig werden für die angelegten Kunden eindeutige Identifizierungsnummern (Kundennummern) vergeben. Mit der Anlage eines Kunden im Kundenmanagementsystem erhält der Kunde die Möglichkeit, bei B2B-Unternehmen einzukaufen. Das Kundenmanagementsystem sorgt dafür, dass ein Kunde dezentral, sprich von verschiedenen Marktstandorten (weltweit), erfasst und verwaltet werden kann. Dieser Prozess geschieht automatisiert, daher werden folgende Prozesse zusätzlich angestoßen:

Prozess 1: Die neu angelegten Kunden müssen in ein angebundenes Buchhaltungsprogramm, genauer: SAP-System (Applikation 4), übertragen und dort als Debitoren geführt werden. Der Prozess, die Daten zusätzlich in einen eigenen SAP-Mandanten auszulagern, ist ein „Best Practice“-Ansatz aus der Praxis und soll anderen Systemen über eine SAP-Standardschnittstelle diese Daten anbieten. Ein Buchhaltungsprogramm wie SAP-FI (Applikation 5) kann dadurch einfach aus einem zentralen System (Applikation 4) jederzeit Kundenstammdaten importieren.

Prozess 2: Analog dazu erfolgt parallel die Datenübertragung vom Kundenmanagementsystem sowohl zum Kassenmanagementsystem als auch zum Kreditmanagementsystem (Applikation 2). Jeder neu angelegte B2B-Kunde, mit dem Geschäftsbeziehungen eingegangen werden, muss einer Bonitätsprüfung unterzogen werden. Im Scoringverfahren wird die Kreditwürdigkeit eines Kunden überprüft. Dafür wird aus Prozess 2 automatisiert ein Unterprozess 2.1 impliziert. Dazu werden die erfassten Kundenstammdaten über eine Schnittstellenanbindung an das zentrale Kreditmanagementsystem (Applikation 3) transferiert. Kreditmanagementsysteme unterscheiden sich von Unternehmen zu Unternehmen

[53] Das zeigt die Praxiserfahrung aus mehreren Softwareprojekten rund um verteilte betriebswirtschaftliche Softwaresysteme.

in den hinterlegten Verkaufsrichtlinien des jeweiligen B2B-Unternehmens. Das letzte Wort über eine Kreditvergabe liegt aber trotz aller IT-Unterstützung beim Kundenmanager, welcher einem Kunden zugeordnet ist. Die Logik von Kreditmanagementsystemen zu testen, erfordert eine hohe fachliche Expertise im jeweils eingesetzten Kreditsystem. Für das Fallbeispiel dieser Dissertation soll das Kreditmanagementsystem als eine Art Blackbox agieren und auf der Basis von Kundendaten jeweils ein Kreditlimit errechnen. Die Berechnung des Kreditlimits selbst erfolgt mit Hilfe von angebundenen, externen Wirtschaftsauskunfteien, wie bspw. SCHUFA[54], CEG[55] oder auch Creditreform. Die Ergebnisse werden sowohl an das Kassenmanagementsystem als auch an das Kundenmanagementsystem gemeldet. Weiterhin werden die Kundenmanager, die jeweils für einen Kunden verantwortlich sind, informiert.

Der Nachteil ist, dass Unterprozess 2.1 eine ständige Onlineverbindung zwischen Kassensystemen, Kreditmanagementsystemen und externen Auskunfteien erfordert. Zusätzlich müssen diese Informationen immer sehr zeitnah den Kundenmanagern zur Verfügung gestellt werden. Um die Kunden nicht unnötig an Kassenterminals warten zu lassen, werden über den Tag verteilt mehrere Zeiträume definiert, wonach ein Austausch von Kreditinformationen zwischen Kreditmanagementsystemen und Kassensystemen erfolgt. So ist sichergestellt, dass immer und tagesaktuell die Kreditverfügung des Kunden in den Kassensystemen zur Verfügung steht. Kassensysteme verfügen über eine Funktion, welche Kreditinformationen über einen Kunden speichern kann. Darum können die ermittelten Kreditlimits direkt an das interne Kreditsystem, welches sich im Kassensystem (Applikation 2) befindet, übertragen werden und dadurch wird eine Standleitung zum Kreditmanagementsystem nicht mehr notwendig.

Ein weiterer Prozess ist die zeitgleiche Reduzierung der verkauften Waren vom Bestand im Warenlager. Dazu existieren zwischen Kassenmanagementsystem und Warenwirtschaftssystem (Applikation 7) mehrere Schnittstellen, von denen zumindest eine die von der Kasse gemeldeten Bestandsreduzierungen von Artikeln direkt vom Lagerbestand abzieht. Warenwirtschaftssysteme sind wiederum mit weiteren Softwaresystemen verbunden, welche bspw. für die automatische Bestellung von Artikeln verantwortlich sind. Diese Prozesse sind für die Dissertation nicht relevant, daher werden sie an dieser Stelle nicht näher betrachtet.

Die Initialisierung eines B2B-Geschäftsprozesses wird durch den Kauf eines Kunden ausgelöst. Je nach Neu- oder Bestandskunde werden die hier kurz dargestellten Prozesse durchlaufen. Je nach Bonität kann ein Kunde dann aus verschiedenen Zahlungsvarianten auswählen:

- Barzahlung – Ausstellung der Rechnung
- Lastschrifteinzug (EC-Karte, Kreditkartenzahlung)
- Rechnung (Zahlung innerhalb von 30 Tagen)
- Schecks/Gutscheine

[54] Wirtschaftsauskunfteien sind Schutzvereinigungen, die die Bonität von Privat- und Geschäftspersonen beurteilen. Unternehmen verlassen sich zumeist auf die Beurteilung und entscheiden auf der Grundlage ihrer Beurteilungen.

[55] Creditreform Consumer GmbH.

Allein wegen den Anspruchsstellungen gegenüber einem Kunden muss eine Rechnung[56] in jedem Fall geschrieben werden. Unabhängig von einer Zahlungsvariante genießen bei B2B-Unternehmen sogenannte Großkunden[57] gesonderte Konditionen und erhalten Rabatte auf Waren. Der Fokus beim Testen muss sowohl die Schecks als auch die Gutscheine berücksichtigen, vor allem beim Verbuchen der jeweiligen Posten im Buchhaltungsprogramm SAP-FI. Genau diese verschiedenen Facetten machen diesen Prozess sehr umfangreich und gleichzeitig auch fehleranfällig gegenüber Anpassungen und Veränderungen.

Die Erläuterung und die Demonstration sollen auf zwei Ebenen erfolgen; zunächst in einer abstrakten Darstellung und aufbauend auf dieser Darstellung sollen dann die an der Lastschrift beteiligten Prozesse und Softwaresysteme skizziert werden. Der Lastschriftprozess berücksichtigt alle grundlegenden und kritischen Geschäftsprozesse, die in der IT-gestützten Zahlungsabwicklung auftreten können. Deshalb eignet sich dieser Prozess besonders als Fallbeispiel und gleichzeitig als „Best Practice"-Demonstration.

3.1.4 Prozess-Fallbeispiel: Bezahlung per Lastschrift

Annahme: Ein Kunde kauft regelmäßig ein und erhält vom Kreditmanagementsystem, abhängig von seinen bisher erbrachten Umsätzen sowie von seinem Zahlungsverhalten, ein errechnetes Kreditlimit in Höhe von 50.000 Euro. Die eingekauften Produkte müssen nach Rechnungserstellung fristgemäß nach 30 Tagen beglichen werden, so dass der Kunde die eingekauften Produkte zunächst auf „Vertrauen" einkauft und das Unternehmen, welches die Produkte verkauft, dem Kunden einen zinslosen Kredit von 30 Tagen gewährt. Das Vertrauen kann durch die Zuhilfenahme von Kreditmanagementsystemen und durch eine Bonitätsprüfung abgesichert werden.

Zahlungsvariante: Die Zahlungsart Lastschrift kann als Zahlungsvariante am Kassensystem durch einen Kundenmanager ausgewählt werden, dadurch werden mehrere Prozesse sowohl im Kassensystem als auch in den angebundenen Softwaresystemen ausgelöst.

Sind die Produkte mittels Kassenperipherie eingescannt und verarbeitet worden, so werden anschließend die Zahlungsvariante Lastschrift und eine Rechnungserstellung gewählt. Eine Rechnung wird unabhängig von der Zahlvariante am Kassensystem ausgedruckt und dem Kunden sowohl direkt im Markt als auch bei einer Lieferung per Lieferservice ausgehändigt. Mit dem Ausdruck der Rechnung ist der Kassenprozess zunächst abgeschlossen. Das Kassensystem sichert anschließend die Umsatzdaten und gleichzeitig die Kundendaten inklusive der Zahlungsvariante (gekennzeichnet mit einer Sequenznummer). Eine Sequenznummer kann bspw. „001" für Lastschriftzahlung sein oder auch „002" für Barzahlungen usw. Die Kennzeichnung mittels Sequenznummern dient dem Buchhaltungsprogramm SAP-FI bei der Unterscheidung, welche Zahlungsvariante ausgewählt worden ist; dies bestimmt gleichzeitig, welche Folgebearbeitung des Prozesses erfolgen soll. Ein Buchhaltungsprogramm wie SAP-FI kann anhand der von der Kasse übergebenen Daten und der Sequenznummer entscheiden, wie die gemeldeten Umsatzdaten zu einer Kundennummer

[56] BGB § 286.
[57] Großkunden sind Kunden, die regelmäßig zu hohen Summen, in der Regel ab 500 Euro, einkaufen. Allerdings entscheidet das Unternehmen immer selbst.

im Buchhaltungsprogramm zu verarbeiten sind. Zunächst erfolgt die Zuordnung der Kundennummer zur korrespondierenden Debitorennummer im Buchhaltungssystem. Dabei werden Barzahlungen bspw. direkt verbucht. Lastschriftzahlungen können erst dann verbucht werden, wenn das Bankeneinzugsverfahren (Bankenprozess im Bankenmanagementsystem) erfolgreich durchlaufen wurde und die offenen Posten eines Kunden als ausgeglichen gemeldet werden.

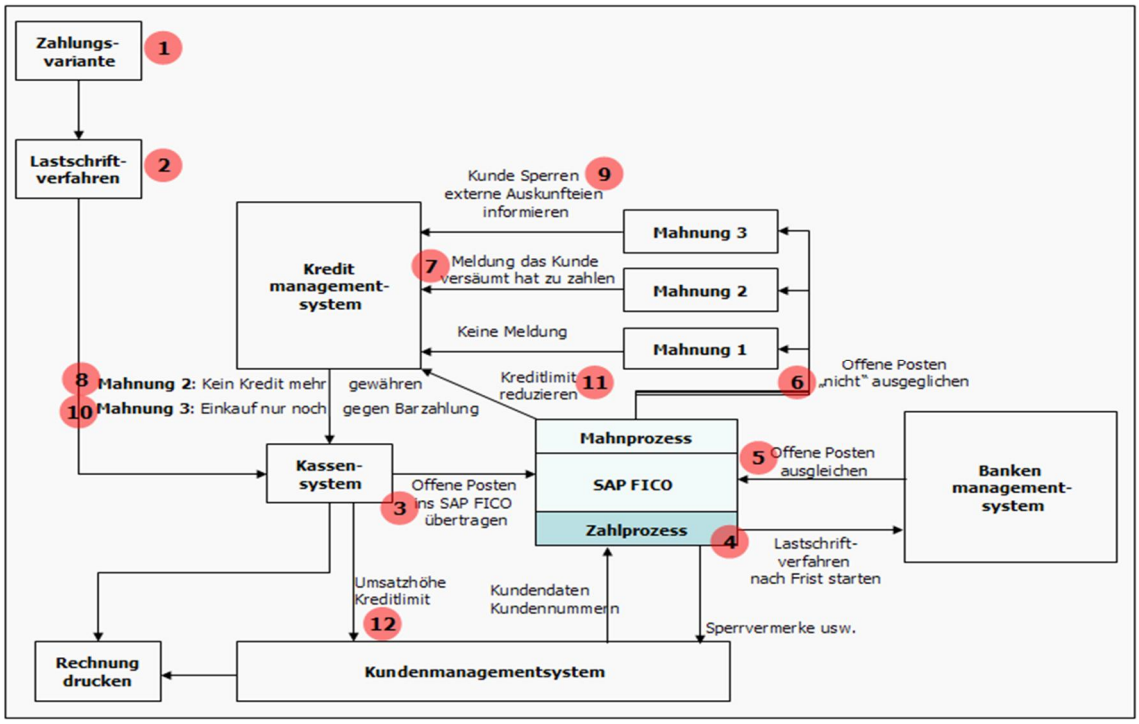


Abbildung 19: Prozessablauf Lastschrift, angelehnt an Fußnote 49

Zahlläufe werden täglich mit SAP-FI durchgeführt, um fällige Verbindlichkeiten auszugleichen. Im Zahllaufprogramm werden zunächst einmal Vorschläge mit sämtlichen Verbindlichkeiten erstellt. In einem weiteren Schritt erfolgt eine Plausibilitätsüberprüfung, hier kann es passieren, dass es bei Unstimmigkeiten zwischen dem Betrag, welcher eingezogen werden soll, und dem vom Buchhaltungsprogramm SAP-FI gemeldeten offenen Posten Differenzen gibt. Existieren keine Unstimmigkeiten, läuft der Prozess weiter und startet das Bankenmanagementsystem, welches für die jeweiligen Bankentransaktionen verantwortlich ist. Dabei wird der Bankensoftware die Liste der einzuziehenden offenen Posten mit jeweils Kundennummer, Debitorennummer, offenem Betrag und Kontoinformationen übergeben. Die Liste wird, beginnend mit der ersten Stelle, abgearbeitet und zu jedem Kunden werden nach erfolgreichem Mandatseinzug Vermerke (erfolgreich, nicht erfolgreich) hinterlegt. Einzugsprozesse, die auf Fehlermeldungen treffen, werden klassifiziert, bspw. mündet die Fehlermeldung „mangelnde Deckung oder Kontosperrung“ in den Mahnlaufprozess. Fehlermeldungen mit „falschen Kontoinformationen“ werden an den Zahllaufprozess zur weiteren Klärung zurückgesendet. Nach jedem Einzugsprozess wird ein gesamtheitlicher elektronischer Kontoauszug erstellt und mit den offenen Posten verglichen.

3.1.5 Zahlprozess im Detail

Der Zahlprozess besteht aus mehreren Aktivitäten, was das Testen dieses Prozesses in der Praxis sehr aufwendig macht. Auch werden vom Zahlprozess korrespondierende Prozesse, wie der Lastschriftprozess und der Mahnprozess, ausgelöst. Die Abbildung illustriert im Groben den Prozessverlauf bei einem Zahlprozess, ohne konkret auf die einzelnen Prozessabläufe einzugehen (Gegenstand in Kapitel 4).

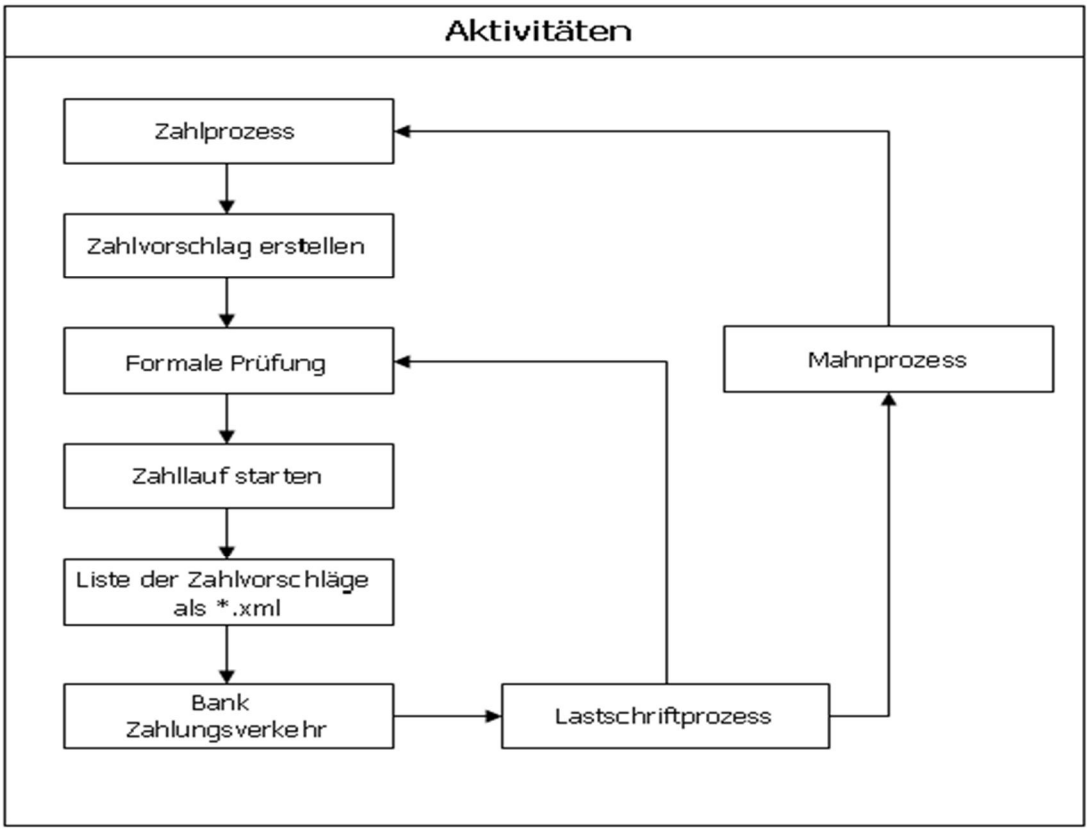


Abbildung 20: Zahlprozess in betrieblichen Informationssystemen, angelehnt an [142]

Die Abbildung demonstriert einen kleinen Ausschnitt eines Prozessablaufs aus einem Gesamtprozess. Folgende Aktivitäten müssen nach dem Zahlprozess ausgeführt werden:

- Bankensoftware muss einen elektronischen Kontoauszug erstellen und auf Basis dieses Auszugs eine Prüfung vornehmen, ob das korrekte Konto belastet worden ist und der korrekte offene Rechnungsbetrag eingezogen wurde. Weiter muss geprüft werden, ob die Mandate zur Prüfung durch einen Fachbereich freigegeben werden müssen oder ob die einzuziehenden Beträge mit den Rechnungsbeträgen übereinstimmen.
- Entspricht der eingezogene Betrag dem Rechnungsbetrag, so werden die offenen Posten in der Buchhaltung als ausgeglichen vermerkt und gleichzeitig verbucht. In einem weiteren Schritt werden sowohl das Kreditmanagementsystem, das Kundenmanagementsystem als auch das Kassensystem über das ausgeglichene Kundenkonto informiert.
- Entspricht der eingezogene Betrag nicht dem Rechnungsbetrag, so werden diese Mandate einem Fachbereich, i.d.R. einem Buchhalter, für eine weitere Prüfung zur Verfügung gestellt. Buchhalter prüfen und informieren die Kunden über diesen

Vorgang. Regulierungen werden an dieser Stelle in Absprache mit dem Kunden durchgeführt.

- Konnte ein Rechnungsbetrag aufgrund eines nicht gedeckten Kontos nicht eingezogen werden, so werden automatisch aus dem Zahlprozess heraus Mahnprozesse ausgelöst. Mahnprozesse bestehen wiederum aus mehreren voneinander abhängigen Aktivitäten.

3.1.6 Mahnprozess im Detail

Der Mahnprozess folgt analog dem Zahlprozess und ist auch in mehrere Aktivitäten unterteilt. Mahnprozesse werden genau dann ausgelöst, wenn die Frist für den Lastschrifteinzug erreicht wurde (30 Tage nach Rechnungsausstellung) und die Zahlung trotz erfolgreicher Aktivierung der Zahlprozesse nicht erfolgen konnte.

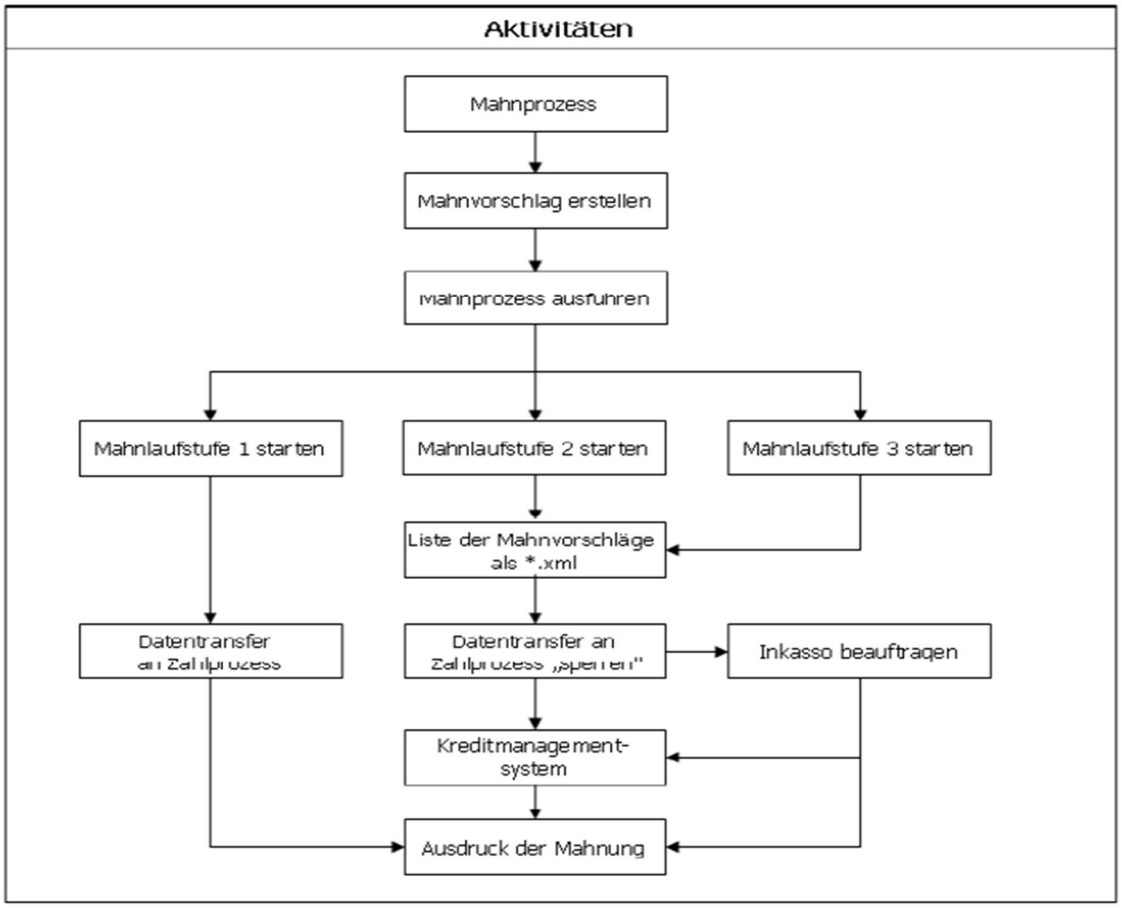


Abbildung 21: Mahnprozess, angelehnt an [142]

Mahnprozesse bestehen in der Regel[58] aus einem dreistufigen Mahnstufenmodell. Dabei fungiert die erste Mahnstufe als Zahlungserinnerung und sollte die Kreditwürdigkeit des Kunden nicht in Frage stellen. Die zweite Mahnstufe informiert die Kreditmanagementsysteme über das säumige Verhalten des jeweiligen Kunden, dadurch werden dem Kunden

[58] Standardmahnprozess, an dem sich alle Großunternehmen orientieren und der auch von Wirtschaftsauskunfteien unterstützt wird. Nur wenn alle Mahnstufen nicht zur Zahlung der offenen Rechnung führen, werden Wirtschaftsauskunfteien darüber informiert und es erfolgt ein dementsprechender Eintrag zu einem Kunden (Unternehmen).

keine weiteren Kredite mehr gewährt. Durch die Zahlsperre wird der Kunde für Zahlungen auf Rechnung oder Kredit gesperrt. Die dritte Mahnstufe führt zur völligen Sperrung des Kunden (Ausnahme Barzahlung), zusätzlich werden Vertragspartner, bspw. Wirtschaftsauskunfteien wie SCHUFA, CEG, Creditreform usw., über das säumige Zahlungsverhalten des Kunden informiert. Die dritte Mahnstufe ist mit einem gerichtlichen Mahnprozess gekoppelt, welchen i.d.R. Inkassounternehmen (bspw. Creditreform) übernehmen. Insgesamt durchlaufen SAP-Prozesse generell immer viele Stufen – wie etwa das Anlegen eines Debitors, das Transferieren von Kassendaten mittels Datenfiles, Zahlläufe oder die Hauptbuchüberleitung. Für den Test dieser Geschäftsprozesse ist es sehr wichtig, bei jedem Testschritt alle Datenkonstellationen zu berücksichtigen und doch das Testziel nicht aus den Augen zu verlieren. Genau diese Prozesse und die korrespondierenden Testdatenkombinationen machen das Testen solcher Geschäftsprozesse sehr umfangreich. Zusätzlich müssen die Schritte in den einzelnen Transaktionen so aufeinander abgestimmt sein, dass die richtigen Daten für den Test erzeugt werden.

3.2 Identifizierte Probleme: Testen der IT-gestützten Zahlungsabwicklung – Fehlerpotentiale und die Notwendigkeit von Tests

Aus den Geschäftsprozessen, die in den Abbildungen 19 bis 21 dargestellt und kurz erläutert wurden, lassen sich für den Test dieser Prozesse mehrere Problemstellungen identifizieren.

Eine der Problemstellungen sind die verschiedenen eingesetzten Softwaresysteme und die unterschiedlichen Produkte. Es handelt sich um eigenständige Softwaresysteme, die in Unternehmen unter verschiedenen Unternehmensorganisationen geführt werden [159]. Die Folge sind heterogene Geschäftsprozesse und komplexe IT-Systemlandschaften, welche Intransparenz und meist auch einen erhöhten Aufwand bei der Steuerung des End-to-End-Geschäftsprozesses erfordern [160]. Auch werden diese Softwaresysteme unter unterschiedlichen Releasezyklen geführt und unterscheiden sich in den Testphasen, so dass jedes Produkt für sich eine eigene Testphase benötigt. Getrieben durch den hohen Wettbewerb stehen B2B-Unternehmen unter einem enormen Zeitdruck, dieser wirkt sich auch auf die Softwaresysteme im Unternehmen aus. So werden die Softwareauslieferungszeiten und dadurch auch die Testphasen für die jeweiligen Softwareprodukte immer kürzer. So zeigt die Praxis, dass in vielen Softwareprojekten die Testphasen für den Kundenabnahmetest kaum drei Wochen übersteigen.

Eine Studie von Steria Mummert Consulting[59] zeigt, dass eine unzureichende Testphase in Softwareprojekten einer der Hauptgründe des Scheiterns oder der nicht planmäßigen Auslieferung bei IT-Projekten ist. Gleichzeitig erhöhen sich die Komplexität und auch die Funktionalitäten von Softwaresystemen, welche durch Tests sichergestellt werden sollen. Je höher die Anzahl der Softwaresysteme, die für die Abarbeitung eines Geschäftsprozesses notwendig sind, desto höher ist die Anzahl der Testfälle, die den reibungslosen Ablauf dieser Geschäftsprozesse sicherstellen sollen.

[59] http://www.steria.com/de/newsroom/downloads/publikationen/studien/.

3.2.1 Problem der Höhe und Anzahl von durchzuführenden Regressionstestfällen

Die Anzahl der durchzuführenden Testfälle pro Softwaresystem und das daraus abzuleitende Ressourcen- und Zeitproblem lassen sich am Fallbeispiel dieser Dissertation verdeutlichen. Die Schätzung der hier aufgeführten Testfälle, welche pro Softwaresystem und pro Release auszuführen sind, beruht auf Überlegungen bezüglich der in der Praxis auszuführenden Testfälle bei B2B-Handelsunternehmen [158]. Die Anzahl konnte aufbauend auf Interviews und Wissensaustausch zwischen Testmanagern unterschiedlicher B2B-Unternehmen vergleichbarer Branchen, wie dem Handel, ermittelt werden. Durch Interviews mit Fachbereichen und Analysen [159] bestehender Testfälle konnten folgende Testfälle ermittelt werden.

- Für das Kundenmanagementsystem werden für einen vollständigen Regressionstest bei einer 100-prozentigen Prozessüberdeckung für eine vollständige Abdeckung aller verfügbaren Geschäftsprozesse 250 manuelle Testfälle benötigt, welche auch in der Produktivumgebung eingesetzt werden. Diese Testfälle beinhalten auch Schnittstellentests zu angebundenen Softwaresystemen. Die Anzahl der Testfälle ist unabhängig von der eigentlichen, im Unternehmen eingesetzten Softwarelösung, da hier die Prozesssicht als Grundlage für Tests herangezogen wird. In B2B-Unternehmen gilt das Produkt von Siebel als Standardprodukt[60] für Kundenmanagementsysteme. Sowohl Carrefour, die Metro Group als auch die Otto Group setzen Siebel ein.
- Je nach Kassenmanagementsystem unterscheidet sich die Anzahl der Testfälle erheblich. Marktführer[61] bei Kassenhardware ist Wincor Nixdorf. Dadurch ist die Verbindung sowohl zu Kundenmanagementsystemen, zum Buchhaltungsprogramm SAP-FI und zu Warenwirtschaftssystemen gegeben. An die Kassensysteme können noch diverse Peripheriegeräte angeschlossen werden, welche bei der Abwicklung (Produkte scannen) unterstützen sollen, wie Barcodelesegerät, Scanner und Drucker für das Drucken der Rechnungen. Für Regressionstests werden 800 Testfälle benötigt. Dazu gehören wiederum Schnittstellentests zu angebundenen Systemen sowie Peripherietests im Labor.[62]
- Das Kreditmanagementsystem ist ein etwas spezielleres Softwaresystem, welches sehr umfangreich getestet werden muss. Speziell deswegen, weil jedes Unternehmen andere Finanzrichtlinien gegenüber seinen Kunden festlegt. Für alle möglichen Kombinationen der kritischsten Geschäftsprozesse im Kreditmanagementsystem sind 1200 manuelle Testfälle vorgesehen. Komplex und aufwendig sind die Tests aufgrund der verschiedenen Äquivalenzklassen und aufgrund der zu jedem Test zu erzeugenden Ampeln.[63] Diese sind wichtig, um die verschiedenen Kreditlimits, ge-

[60] www.siebel.com.
[61] http://www.wincor-nixdorf.com/internet/cae/servlet/contentblob/477440/publicationFile/16265/WincorVision2_2004.pdf.
[62] Im Labor werden Kassensysteme nachgebaut und mit allem ausgestattet, wie für den Tagesbetrieb in einem Markt. Nur so kann getestet werden, ob Release oder neue Kassensoftware fehlerfrei funktionieren.
[63] Eine Ampel signalisiert nach den Ampelfarben die Einschätzungen von Kreditvergaben aufgrund der eingespielten Daten eines Kunden.

bunden an die Umsatzhöhe usw., zu ermitteln. Schnittstellentests zu externen Systemen wie Wirtschaftsauskunfteien kommen noch erschwerend hinzu.

- Für eine vollständige Prozessüberdeckung werden im Rahmen von SAP-FI 2500 Testfälle[64] pro Release benötigt. Die Anzahl der Testfälle ist eine Schätzung, da diese immer abhängig davon ist, wie viele Buchungskreise tatsächlich mit SAP-FI getestet werden müssen. Buchungskreise bilden eine rechtlich selbstständige Gesellschaft im Buchhaltungsprogramm (SAP). So können Unternehmen, die gleichzeitig mehrere Firmen führen wollen, mehrere Buchungskreise in einem SAP-Mandanten einrichten. Mandanten sind wiederum die höchste Hierarchieebene (Organisationseinheit) in einem SAP-System. Bspw. kann eine Holding im Konzern ein Mandant definieren und darunter x Gesellschaften in Buchungskreisen abbilden. Auch das führt dazu, dass SAP-FI-Systeme immer aufwendiger zu testen sind als andere Softwaresysteme.
- Warenwirtschaftssysteme bilden im B2B zentrale Systeme, da sie sicherstellen, dass Warenbestände durch Kunden abgefragt werden können oder rechtzeitige Nachbestellungen von Produkten erfolgen. Die Integration der Warenwirtschaftssysteme ist folglich die wichtigste Schnittstelle sowohl für die Warenwirtschaftssysteme als auch für die Kassensysteme. Ein vollständiger Prozessüberdeckungstest aller Funktionalitäten von Warenwirtschaftssystemen erfordert 400 manuelle Regressionstestfälle. Diese Anzahl bezieht sich nur auf das Warenwirtschaftssystem und auf die für die Dissertation relevanten Schnittstellensysteme (Kasse und DWH). Wobei das DWH hier nur der Vollständigkeit halber aufgeführt wird. Das Testen des DWH ist nicht Gegenstand oder Bestandteil dieser Dissertation.
- Das SAP-FI-Modul und die Bankensoftware (Bankenmanagementsysteme) sind im Prozess eng miteinander verzahnt. Werden losgelöst nur die Bankenmanagementprozesse getestet, so werden 100 Regressionstestfälle für eine vollständige Prozessüberdeckung benötigt.
- Eines der kompliziertesten Softwaresysteme, das aber in dieser Dissertation nicht im Vordergrund steht, ist das Data-Warehouse (DWH), welches die Datenbasis für mehrere Softwaresysteme bildet. Für das Kundenmanagementsystem dient ein DWH als Datenbasis für die Ermittlung der Kundenstatistik (bspw. wie oft ein Kunde welches Produkt und in welcher Anzahl kauft). Auch bei der Kreditlimitberechnung werden Daten vom DWH herangezogen, bspw. kann das DWH als Entscheidungsgrundlage dienen, um bei einem Kunden das Kreditlimit zu reduzieren oder zu erhöhen. Für ein DWH-Release müssen ca. 300 manuelle Testfälle im Regressionstest durchgeführt werden.

Die Anzahl der Testfälle (3000) errechnet sich aus der Summe der einzelnen Testfälle der Softwaresysteme (für SAP-FI wird nur der Test für einen Buchungskreis eingerechnet). Vergleichbare Wettbewerber im B2B haben in persönlichen Interviews[65] und Workshops eine ähnliche Anzahl an Testfällen bestätigt, wobei die Testfälle Redundanzen enthielten.

---

[64] Ist ein Mittelwert aus Regressionstestfällen der SAP, wenn alle FI-Prozesse getestet werden müssen. Da Tests in Buchhaltungssystemen unter Umständen auf mehreren Buchungskreisen durchzuführen sind, basiert diese Schätzung genau auf dem Testen von 10 Buchungskreisen. Pro Buchungskreis werden somit 250 Testfälle notwendig.

[65] Jürgen Meheus (Testkoordinator bei Carrefour in London) & Patrick Liang (Testmanager bei Carrefour).

Die Praxis zeigt, dass ca. 10–15 % der Regressionstests redundante Prozesse beinhalten und bei einer Reorganisation der Testfälle wegfallen würden. Aber auch nach einer Bereinigung der Testfälle bleiben die Anzahl der Testfälle sowie der Aufwand, um diese Testfälle bei jeder Änderung an einem Softwaresystem zu wiederholen, sehr hoch.

Bei jeder Anpassung, Änderung oder auch Erweiterung eines der hier vorgestellten Softwaresysteme müssen angebundene Softwaresysteme notgedrungen prüfen, ob durch die Anpassungen die Funktionalitäten der angebundenen Elemente nicht negativ beeinträchtigt wurden (Regressionstest).

3.2.2 Problem der klassischen Testautomatisierung (Capture & Replay) bei End-to-End-Tests verteilter BIS

Die hohe Anzahl an Testfällen kann in der Ausführung durch die Testautomatisierung (Capture & Replay) beschleunigt werden. Klassische Testautomatisierungsansätze, die in der Praxis auch heute schon Anwendung finden, sind aber sehr anfällig gegenüber Änderungen in den einzelnen Softwaresystemen [119]. Die Anpassungen, die dann durch die Änderungen der Softwaresysteme an den Testskripten vorgenommen werden müssten, gleichen einer Neuentwicklung von Testskripten [77]. Wenn bei Änderungen in einer Software ein Testskript neu angepasst werden muss, lohnt sich eine Testautomation in großen Softwareprojekten nicht [154]. Praxiserfahrungen und Evaluierungen der klassischen Testautomation in betrieblichen Anwendungssystemen[66] waren auch Gegenstand einer Evaluierung und Ausarbeitung in [77], welche der Autor dieser Dissertation zusammen mit einem Experten der Testautomation der Metro Group verfasste. Das Ergebnis dieser Ausarbeitung ist eine kritische Bewertung der heutigen Testautomationsansätze, welche sich auf (Capture & Replay-) Automationsansätze beschränkt und diese für verteilte betriebliche Informationssysteme als unzureichend erklärt.

Allerdings stellen nicht nur die umfangreichen Prozesse, sondern auch das fehlende Prozessverständnis der einzelnen Tester ein Problem dar. Jeder Tester kennt sich jeweils mit seiner zu testenden Applikation aus, aber die End-to-End-Prozesssicht fehlt [159]. Um End-to-End-Geschäftsprozesse zu testen, müssen Tester eine hohe fachliche Prozessexpertise haben, welche applikationsübergreifend die einzelnen Prozesse zu einem Gesamtprozess zusammenfügt.

Das Ziel ist es nicht, Fehlerfreiheit [1] zu erreichen, sondern sicherzustellen, dass Fehler in der Software erkannt und behoben werden können.

3.2.3 Fehlerquellen und die Auswirkungen auf Unternehmen

Beispiel: Der Kunde (im B2B ein Unternehmen), der vom Kreditmanagementsystem ein Kreditlimit von 50.000 Euro erhielt (Beispiel aus 3.1.4), kauft jetzt Waren in Höhe von 20.000 Euro ein. Im Zeitraum bis zur Begleichung (30 Tage ab Rechnungserstellung) darf der Kunde somit noch für 30.000 Euro über einen Wareneinkaufkredit verfügen. Ent-

[66] ERP-Systeme.

schließt sich ein Kunde, die 30.000 Euro noch vor der Zahlungsfrist in Anspruch zu nehmen, wäre jetzt sein Kreditlimit erreicht.

Ein weiterer Kredit kann diesem Kunden nur erteilt werden, wenn er mindestens eine dieser beiden Rechnungen begleicht. Jetzt begleicht der Kunde fristgerecht seine Rechnung per Lastschrifteinzug, somit müsste der Zahlprozess in Kooperation mit dem Lastschriftprozess dafür sorgen, dass seine offenen Salden im Buchhaltungsprogramm SAP-FI aktualisiert werden und das neue Kreditlimit berechnet wird. Gleichzeitig müssen die angebundenen Kassenmanagementsysteme und die Kundenmanagementsysteme mit den neuen Kreditlimits versorgt werden.

Fehlerquellen können Schnittstellen zwischen Buchhaltungsprogramm und Kreditmanagementsystem bzw. Kassensystem sein. Auch kann es sein, dass eine Schnittstelle falsche Daten oder Datenformate transferiert. Auswirkungen können dann sein, dass die soeben beglichene Rechnung eines Kunden in der Buchhaltung immer noch als offener Posten geführt wird. Das wiederum hat Auswirkungen auf die beiden angebundenen Systeme für das Kreditmanagement und das Kassenmanagement. So kann es im schlimmsten Fall dazu kommen, dass ein Kunde trotz fristgerechter Zahlung und Kreditverfügung keine Produkte mehr auf Rechnung oder Kredit kaufen darf. Hier würde ein Fehler im Prozessablauf dafür sorgen, dass die beglichenen Posten nicht im SAP zurückgemeldet und dort als beglichen vermerkt werden.

Allerdings ist auch die umgekehrte Situation denkbar, in der einem Unternehmen monetärer Schaden entstehen kann, indem das System Kunden Kredite gewährt, obwohl diese Kunden evtl. schon längst durch externe Wirtschaftsauskunfteien gesperrt sein sollten. Dies kann passieren, wenn der vollautomatisierte Mahnprozess nicht fehlerfrei funktioniert.

In heutigen Unternehmen werden die Buchhaltungssysteme wie SAP nicht mehr losgelöst von anderen Softwaresystemen genutzt [8], [33], [142]. Sie werden als Teil einer Systeminfrastruktur integriert und übernehmen eine Teilfunktion in einem Geschäftsprozess. Auch werden die hier kurz skizzierten Prozesse des Zahl- und Mahnlaufs täglich in verschiedensten Varianten mit mehreren Tausenden Lastschriftmandaten ausgeführt. In diesem Abschnitt wurden die Komplexität und die verschiedensten Variationen dieser Geschäftsprozesse erläutert, welche das vollständige Testen dieser kritischen Prozesse in einem kurzen Zeitraum erschweren. Alle bisher genannten Probleme können jetzt zusammengefasst werden:

- Bei jeder Änderung eines Softwaresystems in der Systemlandschaft müssen alle anderen Softwaresysteme regressiv ihre Prozessketten testen, um sicherzustellen, dass Änderungen an der Software keine negativen Auswirkungen auf die angebundenen Softwaresysteme haben.
- Dadurch können kritische Prozesse trotz Priorisierungen nicht alle während einer Testphase getestet werden, was in der Produktivumgebung das Risiko für Fehler erhöhen kann.
- Die Anzahl der auszuführenden Testfälle in den einzelnen Softwaresystemen (inklusive Schnittstellentests) ist so hoch, dass diese manuell in keinem vertretbaren Aufwand während einer Testphase durchzuführen sind.

- Testautomatisierungen in den einzelnen Softwaresystemen führen nicht zum gewünschten Effekt, da Anpassungen an der Software Anpassungen in den Testskripten nach sich ziehen. Die Testautomatisierung kann aber Routineaufgaben übernehmen, wie das Anlegen von Testdaten.
- Es sind bisher keine End-to-End-Testprozesse möglich, da den einzelnen Testern meistens auch das Testverständnis für andere Produkte fehlt. Wichtig ist neben den einzelnen Tests in den Softwaresystemen aber die Prozesssicht, diese muss einwandfrei funktionieren.

Mit Hilfe der MBT-Methode, gepaart mit Skripttechnologien der Testautomatisierung, sollen diese Probleme reduziert werden. So erfolgen die Testlogiken mittels Funktionsbibliotheken und die Testschritte mittels Testmodellen. Das Ziel ist es, nicht nur Testmodelle so zu entwickeln, dass Testfälle daraus generiert werden können, sondern auch zu erreichen, dass sich die End-to-End-Geschäftsprozesssicht in den Vordergrund drängt. So kann jede Veränderung in einem Softwaresystem gleichzeitig eine Anpassung am Modell notwendig machen. Der Testfallgenerator trägt die Verantwortung für die Generierung der Deltatestfälle. Das automatisierte Testen soll in einem letzten Schritt dafür sorgen, dass Testfälle mehrmals pro Release ausgeführt werden können. Damit aber so ein Testvorgehen entwickelt werden kann, ist es notwendig, eine Testsystemlandschaft aufzubauen, welche genau diese zu entwickelnde Methode unterstützt.

3.3 Für den Lösungsansatz erforderliche Testsysteminfrastruktur

Um die vorgestellten Geschäftsprozesse überhaupt prozessieren und verarbeiten zu können, müssen Systeminfrastrukturen aufgebaut werden, die es erlauben, einen End-to-End-Geschäftsprozess von der Entstehung bis zum Abschluss zu unterstützen [155]. Die beiden vorausgegangenen Abschnitte beschreiben sowohl die Geschäftsprozessabarbeitung als auch die resultierenden Probleme beim Testen dieser verteilten Prozesse. Jetzt erfolgt ein Einblick in die benötigte IT-Systeminfrastruktur [159], um erfolgreich Geschäftsprozesse prozessieren und testen zu können. Aus der Prozesssicht unterscheiden sich Unternehmen kaum bei der Vorgehensweise, z. B. bei der IT-gestützten Zahlungsabwicklung. Diese buchhalterischen Prozesse unterscheiden sich allerdings in der eingesetzten Softwarelösung und in der IT-Systemlandschaft. Unternehmen setzen heute vielmehr auf vollautomatisierte Prozesse, mit dem Vorteil, dass Geschäftsprozesse schneller bearbeitet werden können [155]. Der Nachteil ist, dass dafür meistens komplizierte IT-Systemlandschaften vonnöten sind, welche aus mehreren heterogenen Systemen bestehen können. Diese heterogenen Systeme müssen so miteinander harmonisiert werden, dass Schnittstellen miteinander kommunizieren können. Basierend auf Abschnitt 3.1 werden jetzt analog zum „Best Practice“-Fallbeispiel kurz die beteiligten Softwaresysteme skizziert und parallel dazu wird eine „Best Practice“-Architektur vorgestellt, an welcher das Fallbeispiel dieser Dissertation demonstriert werden soll. „Best Practice“ darum, weil diese IT-Systemlandschaft sich in der Praxis für die Abarbeitung von End-to-End-Prozessen verteilter betrieblicher Anwendungssysteme bewährt hat, gleichzeitig können andere Unternehmen diese Architektur als Referenz nutzen, um Geschäftsprozesse effizienter und effektiver zu gestalten.

3.3.1 Kundenmanagementsysteme

Für Unternehmen, die wiederum Unternehmen als Kunden haben, sind Kundenmanagementsysteme unverzichtbare Systeme und gleichzeitig auch Instrumente. In diesem Softwaresystem werden sämtliche Kundeninformationen erfasst und verwaltet. Weiter bieten diese Systeme Auswertungsmöglichkeiten, wie Statistiken zum Kaufverhalten von Kunden. Neben den Kundeninformationen werden im Kundenmanagementsystem auch Kreditverfügungen und Rechnungen aus Einkäufen hinterlegt.

3.3.2 Kreditmanagementsystem

Unternehmen möchten gerne wissen, mit welchen Unternehmen sie Geschäftsbeziehungen eingehen. Zur Klärung dieser Frage gehören auch Bonitätsprüfungen der Geschäftspartner untereinander. In erster Linie schützen sich Unternehmen vor Kunden (Unternehmen), die auf Kredit eingekaufte Produkte oder auch Dienstleistungen nicht bezahlen können. Kreditmanagementsysteme sind zusätzlich online mit Wirtschaftsauskunfteien verbunden und können ständig Kundeninformationen abrufen. Allein die SCHUFA speichert Daten von mehr als 1,5 Millionen im Handelsregister eingetragenen Unternehmen. Beim Einsatz von Softwaresystemen können sowohl SAP-Lösungen als auch explizit auf Kreditsysteme spezialisierte Softwarelösungen gewählt werden. Wobei jedes Unternehmen sich gezielt gegen empfindliche Zahlungsausfälle auf der Kundenseite schützen möchte und dadurch Kreditsysteme heute zum Standard in einer Systemlandschaft nicht nur von B2B-, sondern auch von B2C-Unternehmen gehören. In jedem Kreditmanagementsystem werden die jeweiligen Unternehmensrichtlinien konfiguriert. Diese beinhalten bspw. Informationen wie die, ab welchem Umsatzvolumen einem Kunden Kredite gewährt werden sollen. Wobei der Kreditabfrageprozess unabhängig vom Unternehmen nach einem Standardprozess abläuft.

So können Neukunden direkt auf ihr Zahlungsverhalten überprüft werden und gleichzeitig können Kreditverfügungen vergeben werden. Kassensysteme in B2B-Handelsunternehmen beherbergen zusätzlich ein internes Kreditsystem, welches die Daten aus dem Kreditmanagementsystem für die Kassensysteme immer tagesaktuell zur Verfügung stellt [156]. Die Gründe hierfür sind, dass Kundenmanager oder aber auch Kassierer an der Kasse direkt auf Informationen zu einem Kunden zugreifen müssen. Werden erst zur Laufzeit die Kreditabfragen gestartet, so kann dies dazu führen, dass Kreditabfragen bspw. lange Zeit in Anspruch nehmen. Deshalb muss sichergestellt werden, dass diese Daten immer mehrmals am Tag vom zentralen Kreditmanagementsystem an die Kassensysteme gesendet werden.

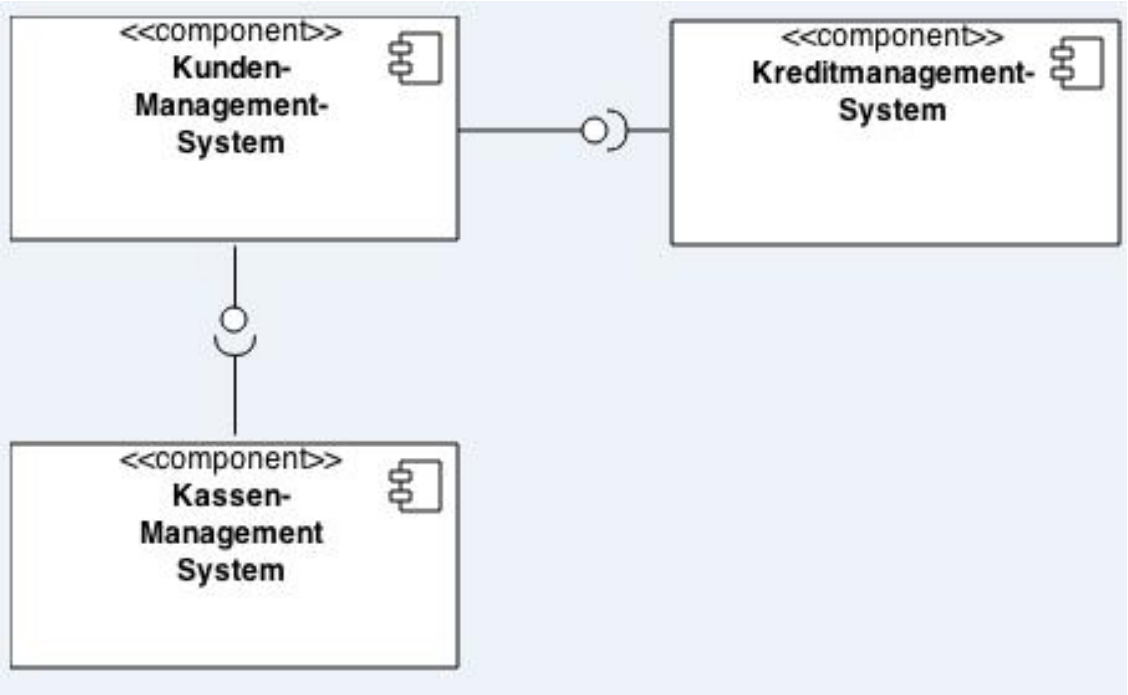


Abbildung 22: Kundensysteme im Verbund, angelehnt an [155] und [142]

Kassensysteme aus verschiedensten Standorten werden an zentrale Kassenserver angebunden. Kassenserver übertragen die erzielten Tagesumsätze in die Buchhaltungssoftware. Kundenmanagementsysteme sind genauso wie die Kassensysteme verteilt und können von Kundenmanagern weltweit in jedem Markt aufgerufen werden. Unternehmen setzen heutzutage an dieser Stelle auf einen automatischen Ablauf ohne Medienbrüche, um dadurch auch Fehlern bei der Datenerfassung vorzubeugen [156]. Die Alternative wäre, die Umsatzdaten durch eine autorisierte Person in der Buchhaltung jeweils in das Buchhaltungssystem buchen zu lassen. Tatsächlich ist das immer noch gelebter Standard bei B2B-Unternehmen, welche über keine Kassensysteme verfügen und Rechnungen über das Kundenmanagementsystem schreiben. Die gemeldeten Tagesumsätze aus dem Kundenmanagementsystem werden dann manuell in die Buchhaltungssysteme transferiert.

3.3.3 Kassensysteme und Warenwirtschaftssystem

Auch bei B2B-Handelsunternehmen sind Kassensysteme heute großen Anforderungen wie Payback, Rabattcoupons usw. ausgesetzt [158]. Heutige moderne Kassensysteme besitzen mehrere Schnittstellen zu geschäftskritischen Softwaresystemen wie Buchhaltungssoftware, Kreditmanagementsystem und Warenwirtschaftssystem [157], [158]. Zusätzlich sind Kassensysteme genauso wie andere Softwaresysteme einem Releasezyklus unterstellt. Kassensysteme sind vor allem bei B2B-Unternehmen mit mehreren anderen Softwaresystemen verbunden. Deshalb ist das Testen von Kassensystemen eine der wichtigsten Aktivitäten und wird zumeist in von den Unternehmen selbst entwickelten Kassenlabors durchgeführt. In B2B-Unternehmen übernehmen die Kassensysteme noch zusätzlich die Aufgabe der elektronischen Rechnungserstellung. In B2B-Unternehmen, die keine Kassensysteme einsetzen, übernehmen Kundenmanagementsysteme die Aufgabe der Rechnungserstellung, bis zum elektronischen Datenaustausch mit dem Rechnungsempfänger.

Warenwirtschaftssysteme spielen in der IT-gestützten Zahlungsabwicklung zwar nur eine untergeordnete Rolle, bilden aber genauso wie die bisher vorgestellten Softwaresysteme eine Teilfunktion im Geschäftsprozess und sind dadurch essentiell wichtig für die erfolgreiche Abarbeitung des Gesamtprozesses. Jeder verkaufte Artikel muss aus dem Bestandslager abgezogen werden. So melden Kassensysteme die verkauften Waren und Produkte über eine Schnittstelle an das Warenwirtschaftssystem, welches dann automatisch den Bestand um die entsprechende Anzahl der Waren und Produkte reduziert.

Auch Warenwirtschaftssysteme sind wiederum mit automatischen Bestellkatalogen verbunden, so dass bei Erreichung einer Mindestanzahl an Artikeln ein automatisierter Prozess im Warenwirtschaftssystem gestartet wird, welcher dafür sorgt, dass dieser Artikel nachbestellt wird.

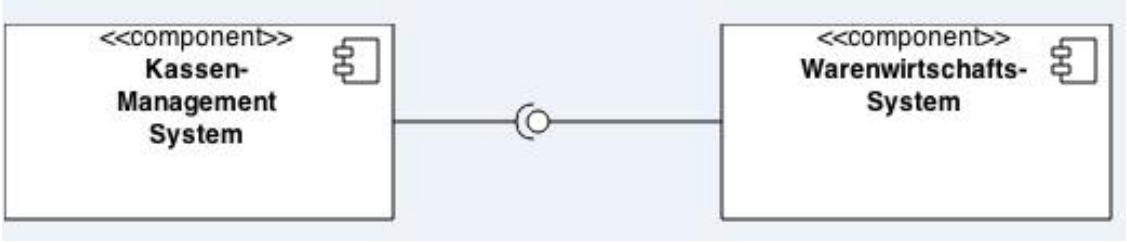


Abbildung 23: Interaktion der Kassensysteme und Warenwirtschaftssysteme

3.3.4 Buchhaltungsprogramm SAP-FI und Bankensysteme

Die gesetzeskonforme und revisionssichere Software von SAP für die Finanzbuchhaltung ist das Herzstück der IT-gestützten Zahlungsabwicklung [142]. Sie unterstützt im Unternehmen alle buchhalterischen Aufgaben, wobei sich viele der hier durchzuführenden Aufgaben und Prozesse automatisieren lassen.

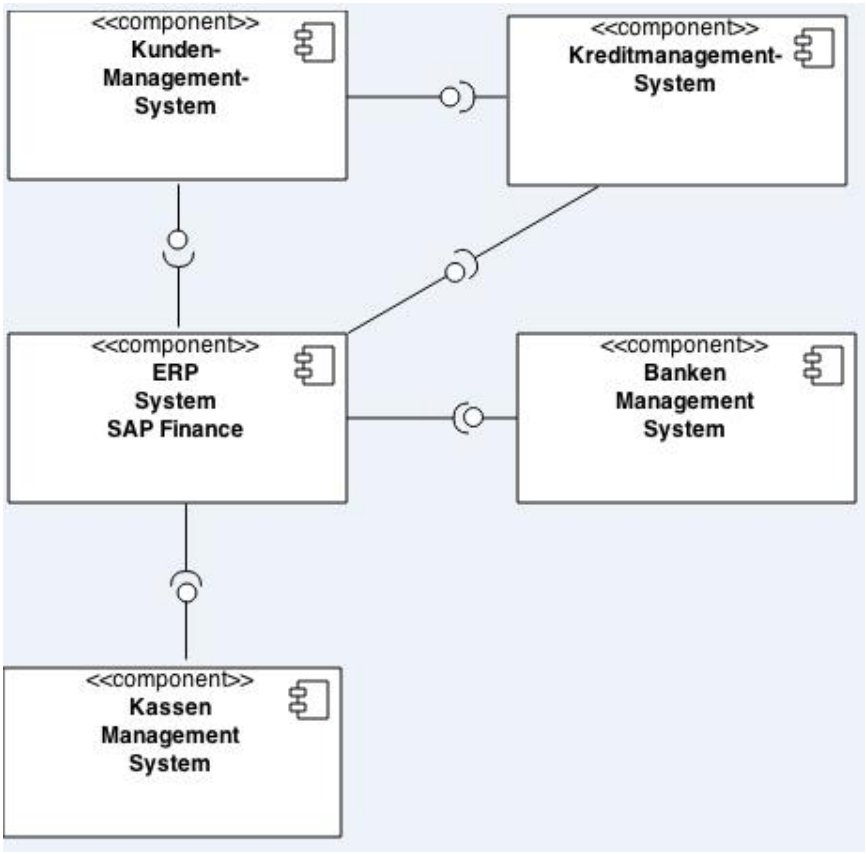


Abbildung 24: Testsysteminfrastruktur, angelehnt an: [155], [158], [142]

Ein Buchhaltungsprogramm wie SAP-FI besteht aus mehreren Untermodulen und bietet durch die offenen Schnittstellen die Möglichkeit, weitere Softwaresysteme anzubinden. Die Finanzbuchhaltung besteht aus mehreren Modulen, wobei die Hauptbuchhaltung mit der Kreditorenbuchhaltung und der Debitorenbuchhaltung das zentrale Element bildet. Die folgende Abbildung illustriert die Integration der an die Finanzbuchhaltung angebundenen Systeme.

So werden bspw. Kassenumsätze in die Buchhaltung übertragen und Debitoren zugeordnet. Die ausgewählten Zahlungsarten bestimmen jeweils die weitere Verarbeitung der Prozesse. Barzahlungen werden sofort in der Debitorenbuchhaltung verbucht. Lastschriftzahlungen werden vorgemerkt, offene Salden dem jeweiligen Debitor zugeordnet und im Zahllaufprozess vermerkt. Analog dazu werden die Zahllaufprogramme so angepasst, dass sie jeweils fristgerecht die Lastschriftmandate einziehen.

3.3.5 Data-Warehouse (DWH)

Das DWH fungiert als Historiendatenbank für alle getätigten Kundentransaktionen und speichert auch redundante Daten. Das Ziel eines DWH ist es, einen umfassenden Überblick über die vorhandenen Daten im Unternehmen zu geben und damit die Entscheidungen der Unternehmensplanung zu unterstützen. Die Integration von Daten aus vielen heterogenen Datenquellen ist ein Kernproblem (welches hier nicht näher betrachtet wird) des DWH. So sind am DWH mehrere Systeme angebunden, wie Kassensysteme, Kreditmanagementsysteme oder auch Warenwirtschaftssysteme, die die Daten eines Kunden speichern. Aggregiert bieten diese Informationen Entscheidern in einem Unternehmen eine

Basisgrundlage, um frühzeitig auf Kundenwünsche zu reagieren, gleichzeitig bieten sie die Möglichkeit, das Kaufverhalten von Kunden und die Absetzung von Artikeln auszuwerten.

Zusammenfassend lässt sich an dieser Stelle festhalten, dass auf der Basis der unter 3.3 skizzierten Systemlandschaften jetzt die konkreten Anforderungen an die einzelnen Softwaresysteme spezifiziert werden können. Kapitel 3.1 beschrieb den IT-gestützten Zahlungsprozess (Debitoren-Prozess) und dessen Ablauf, Kapitel 3.2 skizzierte die in 3.2 zusammengefassten Probleme beim Testen solcher End-to-End-Geschäftsprozesse. Auf der Basis der in Abbildung 17 gezeigten Softwaresysteme und der korrespondierenden Prozesse in Abbildung 18 wurden die Systemlandschaften vorgestellt.

Das Zusammenspiel der einzelnen Softwaresysteme kann nur dann funktionieren, wenn jede Software Kenntnis über die andere Software hat. Das soll die Schnittstellenintegration der Softwaresysteme gewährleisten. Im nächsten Schritt müssen neben der Systemlandschaft, die es zu testen gilt, auch Testwerkzeuge definiert werden, mit welchen das Testen solcher Softwaresysteme strukturiert durchgeführt werden kann.

3.4 Technische Umsetzung des Lösungsansatzes

In diesem Abschnitt soll die Lösungsbeschreibung skizziert werden, dazu gehören neben einer benötigten Testsystemlandschaft auch die notwendigen Testwerkzeuge, die vonnöten sind, um die hier beschriebene Lösung zu realisieren. Die allgemeine Lösungsbeschreibung erfordert analog zur allgemeinen Prozessbeschreibung, welche unter 3.1 vorgestellt wurde, auch eine korrespondierende IT-Systemlandschaft, welche diese Prozesse abbildet. Diese wiederum erfordert für aussagekräftige Tests von End-to-End-Geschäftsprozessen analog zum Produktivbetrieb auch produktnahe Testumgebungen, welche den Tagesbetrieb der Produktivumgebung simulieren können. So soll sichergestellt werden, dass der tatsächliche Praxisbetrieb auch in der Testphase nachgebildet werden kann und dass nicht unter falschen Bedingungen und Voraussetzungen getestet wird.

Beim Aufbau einer Testsystemlandschaft muss zusätzlich beachtet werden, dass Testsysteme so ineinander integriert werden, dass alle Testsysteme einen gleichen Releasestand analog zur Produktivumgebung haben. Nur so ist gewährleistet, dass aussagekräftige und repräsentative Tests durchführbar sind.

Eine zweite Voraussetzung sind Testmanagementtools, welche nicht nur zur Testverwaltung genutzt werden sollen, sondern auch zur Erfassung und gleichzeitig zur Verwaltung von Anforderungen [157]. So werden die Test- und die Anforderungsphase miteinander verknüpft. Neben dem einzusetzenden Testmanagementtool müssen in einem weiteren Schritt auch geeignete Testwerkzeuge für die Testautomatisierung und für die Testfallmodellierung sowie ein Testgenerator ausgewählt werden.

Eine dritte Voraussetzung ist eine klare Definition der einzelnen zu testenden Prozessschritte. Am Fallbeispiel der IT-gestützten Zahlungsabwicklung erfolgt Schritt für Schritt eine Erläuterung zu den einzelnen Prozessschritten und den notwendigen Eingaben und erwarteten Ausgaben. Die Testfallspezifikation bildet in Kapitel 4 die Basis für die spätere Modellierung dieser Testfallspezifikationen. Es müssen Testdaten schon in den Testfällen spezifiziert werden, damit diese bereits bei der Modellierung berücksichtigt werden können.

Dabei muss ein Testfall durchgängig, vollständig sowie zuverlässig sein. Wichtig für End-to-End-Prozesse sind die Zwischenschritte und die Spezifizierung der Testdaten, die zwischen den beteiligten Softwaresystemen transferiert werden sollen.

3.4.1 Die notwendige Testsystemlandschaft für das Testen von End-to-End-Geschäftsprozessen

Das in dieser Dissertation zu demonstrierende Fallbeispiel basiert auf einer allgemeinen Beschreibung einer Testsystemlandschaft mit dem Spezialfall des Demonstrators, welche sich in der Praxis bei vielen B2B-Unternehmen wiederfindet [159], [156], vgl. Abbildung 17. Auf einem Testsystem werden Softwaresysteme, die einen lokalen Charakter besitzen und für das Testen des Gesamtprozesses keine Rolle spielen, bewusst weggelassen. Gleichzeitig zeigt die Praxis, dass Fachbereiche, die letztendlich auch die Anwender dieser Softwaresysteme sind, zusätzlich noch „kleine“ lokale Anwendungen betreiben, mit welchen z. B. Daten aus einer Datenquelle mit Skripten ausgelesen werden können. Auch haben Fachbereiche eigene Testrichtlinien, bspw. um ihre lokal-spezifischen Einstellungen in einer Anwendung zu testen [150]. Aus Kostengründen vermeiden es Unternehmen, lokale Sondereinstellungen auf Testsystemen zu berücksichtigen [140]. Wobei sich die Unternehmen auch hier voneinander unterscheiden, weil jedes Unternehmen das Thema „Testen von Softwaresystemen“ anders gewichtet und priorisiert [138], [158]. Hier muss das Unternehmen entscheiden, mit welchem Risiko es seine Softwaresysteme im Produktivbetrieb betreiben möchte.

Der „Best Practice“-Ansatz für B2B-Debitoren-Prozesse (IT-gestützte Zahlungsabwicklung), welcher sich in der Praxis bewährt hat [160], [150], wird grob in Abbildung 26 illustriert und in den folgenden Unterkapiteln detaillierter erläutert. Gleichzeitig beschreiben die einzelnen Unterabschnitte das Ergebnis, welches sukzessive entstehen soll.

3.4.1.1 Aufbau der Testsysteme

Die Aufteilung der Softwaresysteme ist durch die Historie und durch die Erfahrungen vieler Mitarbeiter über mehrere Jahre hinweg entstanden. Sowohl Einflüsse aus der fachlichen als auch aus der technischen Sicht sind Bestandteil dieser für B2B typischen Systemlandschaft. Mit Hilfe dieser Architektur lassen sich beliebige End-to-End-Geschäftsprozesse von betrieblichen Informationssystemen abbilden und gleichzeitig auch testen. So kann das modellbasierte Testen an dieser Architektur entwickelt werden.

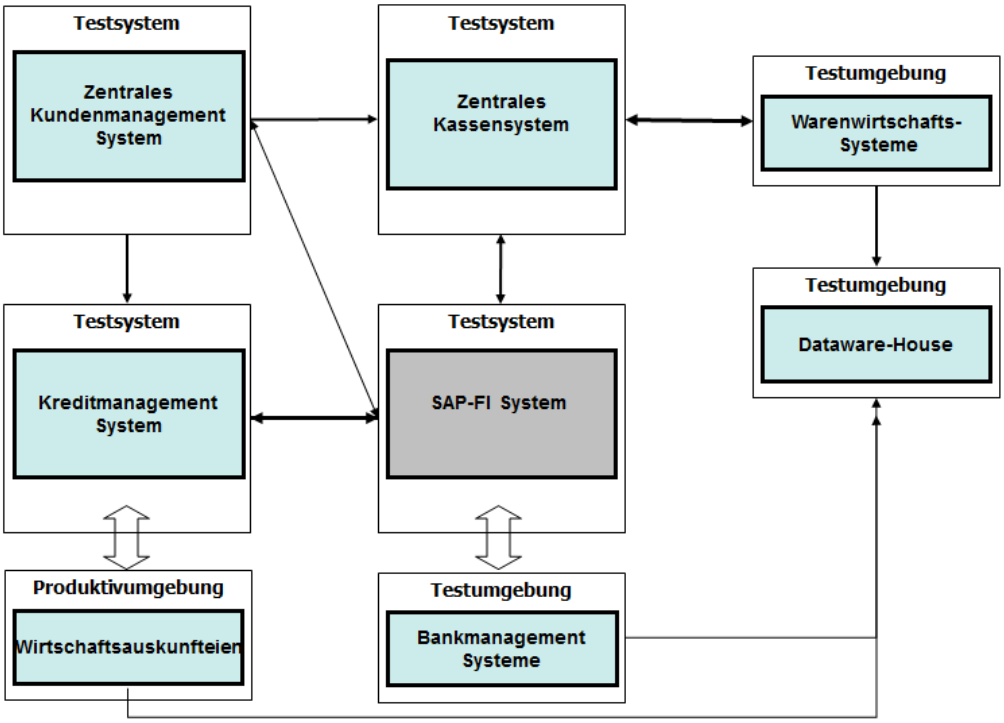


Abbildung 25: Testinfrastruktur zum Testen von B2B-Prozessen

3.4.1.2 Schnittstellen und Verbindungen unter den einzelnen Softwaresystemen

Softwaresysteme wie SAP-FI, CFM, Kassensysteme und Warenwirtschaftssysteme bieten offene Schnittstellen, um eine Anbindung von Drittanbieter-Software zu ermöglichen. SAP ist mit dem CFM über eine SDI-Schnittstelle verbunden. Im SAP-SDI sind alle verfügbaren Schnittstellen von angebundenen Drittsystemen aufgelistet. Die von CFM zu SAP transferierten Daten werden mittels SDI zur weiteren Verarbeitung abgelegt. Ein SDI beinhaltet festgelegte Strukturen und Informationen, die dazu beitragen, die empfangenden Dateien zu unterscheiden und zu identifizieren. SAP bietet im Standard Mechanismen, mit denen dann die erhaltenen Daten weiterverarbeitet werden können. Eine Verbindung zwischen diesen beiden Systemen erfolgt über die Konfiguration des SDI im SAP. Dort wird dem Datenzulieferer CFM die Berechtigung erteilt, Daten in einer definierten Ablagestruktur zu übertragen. Sowohl der Systemname als auch die IP-Adresse des jeweiligen CFM-Servers werden dazu benötigt. Die SDI-Schnittstelle übernimmt gleichzeitig die Adaptierung der Daten, so dass diese mittels SAP verarbeitet werden können.

Sowohl das Kundenmanagementsystem als auch die Kassensysteme sind über jeweilige Firmennetzwerke mit den einzelnen Applikationsservern verbunden. Für das Entwickeln und Aufbauen einer Testumgebung genügt es, mittels CFM den Kassen-Applikationsserver auszuwählen und die beiden Softwaresystemserver miteinander zu verbinden. Täglich werden von den Kassenservern Massendaten an das CFM versendet. Die Datensätze enthalten Aktualisierungen, welche aus angebundenen Systemen an die Kasse gemeldet werden. Die Aktualisierungen werden in fest eingeplanten Perioden an das CFM transferiert. Umgekehrt werden vom CFM Datenaktualisierungen direkt an die Kasse gemeldet.

Das Kreditmanagementsystem ist direkt mit dem CFM verbunden. Jeder neu erfasste Kunde muss unmittelbar einer Bonitätsprüfung unterzogen werden. Aber nicht nur dieser Prozess erzwingt einen Bonitätsprüfungsprozess, sondern auch Änderungen, welche von SAP-FI gemeldet werden, können zu einer neuen Berechnung des Kreditlimits führen. Täglich werden Tausende von Mandaten durch SAP-FI verarbeitet, sprich es werden Lastschrifteinzüge durch den Zahllaufprozess gestartet. Hinter jedem Mandat stehen jeweils Kunden und Kundenstammsätze. So müssen im Beispiel je nach Verlauf des Zahllaufprozesses die Kundenstammdatensätze an das Kreditmanagementsystem zurückgemeldet werden.

Bei den Warenwirtschaftssystemen, welche auch direkt mit den Kassensystemen verbunden sind, verlaufen die Transaktionen ähnlich. So muss jeder verkaufte Artikel / jedes Produkt vom Gesamtbestand reduziert werden und zu jedem Kunden, der eine Anzahl von Produkten eingekauft hat, wird zusammen mit der Rechnung im Hintergrund eine XML-Datei erzeugt, welche die gekauften Artikel enthält. Diese Datei wird an den Applikationsserver des Warenwirtschaftssystems versendet. Ein automatisierter Prozess empfängt die Datei und verarbeitet die enthaltenen Informationen.

Zwischen dem Zahlprozess und der tatsächlichen Einziehung der Lastschriftmandate muss noch ein Bankenmanagementsystem durch eine Online-Verbindung zu den Banken der Kundenkonten aufgebaut werden. Die einzuziehenden offenen Salden werden in Form einer XML-Datei zur Verfügung gestellt und online an die Bank transferiert und gleichzeitig wird der Lastschriftprozess gestartet. Nach dem Abarbeiten des Bankenprozesses generiert das jeweilige Bankenmanagementsystem eine XML-Datei, welche die eingezogenen Zahlungen oder auch die Fehlbuchungen zu dem jeweiligen Kunden enthält. Diese XML-Datei wird von einer sogenannten Batch-Input-Mappe direkt in SAP verarbeitet und es werden alle eingehenden Zahlungen direkt in das Hauptbuch von SAP-FI verbucht.

3.4.1.3 Wiederverwendung von Systemkomponenten und Datenmengen

Ein großer Vorteil ist, dass die hier dargestellten Systemkomponenten und Prozesse wiederverwendbar sind. So kann bspw. ein Testfall, welcher für das Starten und das Anmelden beim CFM-System beschrieben ist, beliebig auf andere Softwaresysteme übertragen werden. So reicht es in der späteren Modellierung aus, anhand eines Softwaresystems die Anmelderoutine einmal zu modellieren. Dieses Modell kann dann als Basis wiederum anderen Testmodellen zur Verfügung gestellt werden. Es müssen an dieser Stelle Parameter hinzugefügt oder der Softwaresystemname muss angepasst werden.

Bei den zu transferierenden Datenmengen zwischen den Applikationen gibt es Unterschiede. So werden die größten Datenmengen zwischen SAP-FI und den Kassensystemen und zwischen SAP-FI und den Kreditmanagementsystemen bewegt. Erwartet wird eine tägliche Verarbeitung von Massendatensätzen zwischen Kassensystemen und SAP-FI. Gleichzeitig werden täglich Mandate über den Zahllaufprozess ausgelöst und müssen verarbeitet werden. Auch müssen die Ergebnisse der Zahlprozesse zu den angebundenen Systemen transferiert werden. Das Kreditmanagementsystem muss bei einer täglichen Abarbeitung von hunderttausend Mandaten die gleiche Anzahl an Kreditlimits neu berechnen und diese so-

wohl an die Kassen- als auch an die CFM-Systeme melden. Zeit ist ein entscheidender Faktor bei der Abarbeitung dieser Prozesse. So existieren in den verschiedenen Unternehmen strikte Zeitpläne, bis wann bspw. die Lastschriftmandate ihre Abarbeitung abgeschlossen haben müssen und wann die Kassensysteme mit aktualisierten Informationen allerspätestens versorgt werden müssen.

Der Vorteil dieser Testsystemlandschaft ist, dass der Aufbau durch ein „Best Practice“ aus dem Produktivbetrieb quasi vorgegeben wurde und für die Testumgebung nur nachgebaut werden musste. Die rot umrandeten Softwaresysteme mussten zusätzlich für die Aufgabenstellung dieser Dissertation angebunden werden. Externe Wirtschaftsauskunfteien während einer Testphase (zu Testzwecken) online über Kundendaten abzufragen oder auch „echte“ Banktransaktionen auf einer Testsystemlandschaft durchzuführen, ist nicht möglich. Es bleibt aber der Zugriff auf Testsysteme, welche Testdaten beinhalten. Zu Testzwecken wurde mit Wirtschaftsauskunfteien eine Verbindung zu deren Testsystemen aufgebaut. Zusätzlich werden für die Simulation der Bankenmanagementprozesse Bankenclients und Benutzerkonten und auch das DWH zu Testzwecken aufgebaut.

Neben der Testsystemlandschaft ist die Testdatenqualität essentiell wichtig für einen erfolgreichen und aussagekräftigen Test. Als Basis werden Daten aus der Produktivumgebung für Testzwecke auf die Testsysteme kopiert. Anschließend müssen diese Testdaten genau analysiert und passend mit Äquivalenzklassen erweitert werden. Die Kombination aus Produktivdaten und Äquivalenzklassen sorgt für einen breiten und hohen Testabdeckungsgrad.[67]

3.4.2 Vorgehensweise MBT

Beim MBT werden Testfälle anhand von Modellen generiert, welche das geforderte Verhalten des zu testenden Softwaresystems abbilden. Zu Beginn des MBT werden die Anforderungsspezifikationen des zu testenden Systems herangezogen und in ein Testmodell überführt. Ein Zwischenschritt besteht darin, die Anforderungsspezifikation in Form eines abstrakten Testfalls zu spezifizieren, im Speziellen für solche komplexen Geschäftsprozesse, wie unter 3.1 beschrieben. Diese dienen dann dem Business-Analyst als Basis und führen zum besseren Verständnis der Modellierung. Die Vorgehensweise:

- Anforderungsspezifikation in einen abstrakten Testfall überführen.
- Abstrakten Testfall in ein konkretes Testmodell überführen.
- Generierung der Testfälle mit Hilfe geeigneter Testwerkzeuge.
- Adaptierung der generierten manuellen Testfälle an ausführbare Testskripte.
- Automatisierte Ausführung der Testskripte inkl. Auswertung der Testergebnisse.

Für die Spezifikation des Modells stehen unterschiedliche Modellierungsnotationen zur Verfügung, wobei BPMN 2.0 (Business Process Modeling Notation) als Modellierungssprache ausgewählt wird. Anhand des erstellten Testmodells werden dann zunächst mittels eines Testfallgenerators manuelle Testfälle generiert. Diese sollen die Abdeckungskriterien, wie bspw. der Pfadüberdeckung oder der Prozessüberdeckung, des Modells erfüllen. Die manuellen Testfälle müssen in einem weiteren Schritt an ausführbare Testskripte adaptiert werden.

[67] Vorlesung „Software Engineering für große Informationssysteme“, TU Wien SS2007, S.57.

3.4.3 Anforderungen an das MBT für betriebliche Informationssysteme

Das Ziel ist es, die hohe Anzahl von auszuführenden Testfällen, welche bei jeder Änderung an einem Softwaresystem notwendig werden, zu beherrschen, um Fehler zu identifizieren [150]. Auch die neu hinzukommenden Funktionalitäten, welche Erweiterungen und Anpassungen an die Softwaresysteme mit sich bringen, werden vielseitiger und erhöhen die Anzahl der auszuführenden Testfälle. Auch die systematische Testfallableitung von einem Testfallmodell zur Anwendung soll dadurch gefördert werden. Ein weiterer Vorteil liegt in der leicht durchzuführenden Wartung. Werden Veränderungen am zu testenden System bzw. an der Anforderungsspezifikation vorgenommen, so können die notwendigen Anpassungen leicht umgesetzt werden. Dazu müssen die betroffenen Stellen des Testmodells lediglich modifiziert werden, was auf Grund der abstrakten Darstellung des Testmodells einfach umzusetzen ist.

Des Weiteren ist es von Vorteil, dass aus einem gleichbleibenden Modell identische Testfälle generiert werden. Auf Grund dessen können die aus jedem Testdurchlauf resultierenden Ergebnisprotokolle effektiv für den Regressionstest verwendet und dadurch Fehler identifiziert werden. Auch die Steigerung des Testabdeckungsgrades und die Effizienz des Testprozesses zu verbessern, ist eine Anforderung an das MBT. Abgesehen von der zu berücksichtigenden Funktionalität von MBT-Testwerkzeugen und der Bedienfreundlichkeit, sollte das MBT-Testwerkzeug in der betroffenen Domäne (Testen verteilter betrieblicher Informationssysteme) einsetzbar sein. Zusätzlich muss ein MBT-Testwerkzeug sich in gängige, am Markt befindliche Testwerkzeuge integrieren lassen. Sowohl die Testfallmodellierung als auch die Testfallgenerierung sollte nach einer strukturierten Vorgehensweise erfolgen. Der Schwerpunkt sollte auf verteilten ERP-Systemen in einer heterogenen Systemumgebung liegen. Konkret sollte für diese Dissertation ein Testwerkzeug ausgewählt werden, welches sich in gängige Testmanagements integrieren lassen kann.

Die Testausführung sollte exakt nachweisbar sein und es sollte eine Dokumentation der einzelnen Testschritte automatisiert erfolgen. Gescheiterte Testfälle (im Fehlerfall) sollten so dokumentiert sein, dass ein Tester diesen Testschritt nachvollziehen und reproduzieren kann.

3.4.4 Auswahl der MBT-Testwerkzeuge

Bei der Auswahl eines geeigneten Testwerkzeugs müssen einige Faktoren berücksichtigt werden, bspw. inwiefern sich die Anforderungen dieser Dissertation mit Hilfe der MBT-Testwerkzeuge abdecken lassen. Oder: Wie lassen sich generell die verschiedenen MBT-Testwerkzeuge in eine etablierte Testumgebung mit schon vorhandenen Testmanagementwerkzeugen implementieren? Dazu gehören auch die Benutzerfreundlichkeit und die Erlernbarkeit dieser Testwerkzeuge. Anders als in vielen bisher vorgestellten MBT-Lösungen [101], [102], [103], agieren beim Testen betrieblicher Informationssysteme keine technisch versierten Ingenieure, sondern es handelt sich vielmehr um eine Zielgruppe, welche wenig IT-Background besitzt.

Um von vornherein die Fachbereiche für einen MBT-Ansatz zu gewinnen, muss als Basis ein verständliches Werkzeug dienen, welches es einem Fachbereich zusammen mit einem Modellierer (Business-Analyst) erlaubt, Anforderungen in Testfallmodelle zu überführen. Erfahrungen aus der bisherigen Praxis[68] zeigen, dass so bereits in einer frühen Phase konzeptionelle Defizite des geplanten Softwaresystems aufgedeckt werden können, die mit „klassischen" Methoden erst sehr spät oder auch gar nicht bemerkt werden, wenn sie nicht zu fehlerhaften Tests führen. Allerdings ist die Modellierung aufwendig und erfordert Expertise in der Modellierung von Geschäftsprozessen.

BPMN erlaubt einen Spagat, mit dem sowohl Fachabteilungen als auch die technischen Umsetzer arbeiten können, und schafft ein durchgängiges Modell. Neben der Modellierungsnotation ist ein Testfallgenerator notwendig, um mit Hilfe von Algorithmen aus den Testmodellen Testfälle zu generieren. Wie im Kapitel 2 „Stand der Technik" schon beschrieben, existieren mehrere MBT-Testwerkzeuge am Markt, wobei diese Dissertation in Kapitel 2 eine Auswahl durchleuchtete.

Durch intensive Recherchen und Interviews mit verschiedenen MBT-Testwerkzeug-Herstellern[69], aber auch mit MBT-Testwerkzeug-Kunden[70], schafften es drei MBT-Testwerkzeuge nach den gestellten Anforderungen an MBT-Testwerkzeuge in die engere Auswahl für den Lösungsansatz dieser Dissertation: Qtronic (RUDI) von der Conformiq GmbH, Tedeso und TestBench von der Imbus AG und der „Innovator for Business Analyst" mit der MBTSuite von der SeppMed GmbH und der MID GmbH.

- Nach einer weiteren Untersuchung vom Autor dieser Dissertation auf die Praxistauglichkeit beim End-to-End-Testen von verteilten BIS stellte sich heraus, dass sowohl die Lösungen von Conformiq als auch von der Imbus AG für die Anforderungen dieser Dissertation nicht ausreichend sind. Die ausschlaggebenden Kriterien, nach denen der Autor vorging, sind:
- Die Modellierungsnotation sollte für Fachbereiche leicht verständlich und lesbar sein. Als Modellierungsnotation kristallisierte sich in vielen Interviews mit Fachbereichen BPMN 2.0 heraus.
- Modellierung von End-to-End-Testprozessen mit Testdatenspezifikationen an den Schnittstellen, wobei es nicht ausreichend ist, eine (wahr/falsch oder ja/nein)-Abfrage durchzuführen, sondern es muss eine Datenabfrage auf Datenformat und Inhalt erfolgen. D. h., in der Modellierung muss es möglich sein, Schnittstellendaten zu spezifizieren, es reicht definitiv nicht aus, eine (ja/nein)- oder (wahr/falsch)-Abfrage zu generieren.
- Die Integration von Testfallgeneratoren in Testmanagementtools und in Testautomatisierungswerkzeugen sollte gegeben sein.

Bei der Modellierung von Tests gab es zwischen den drei verbliebenen Testwerkzeugen Unterschiede. Tedeso setzte u.a. bei der Modellierung von Geschäftsprozessen auf Aktivitätendiagramme. Das gemeinsam von Siemens Corporate Technology und den Testexperten der Imbus AG erarbeitete Produkt „Managed Model Based Testing" stellt eine Kombination aus Siemens' Testgenerator Tedeso und Imbus' Testmanagementtool TestBench

[68] Frank Plaster und Okadis Consulting GmbH – Success Story bei Einführung SAP ERP 6.0.
[69] MBTSuite, SeppMed, Imbus AG mit Tedeso, Conformiq mit Qtronic.
[70] Telefoninterviews: Daimler, Audi, Siemens Health-Care.

dar. Tedeso nutzt hauptsächlich UML als Modellierungsnotation und je nach zu testender Softwarelösung verschiedene UML-Diagramme. Mit Zustandsdiagrammen lassen sich zwar Methoden aus Prozessen identifizieren und auch ableiten, allerdings können damit keine sinnvollen betriebswirtschaftlichen Geschäftsprozesse modelliert werden. Mit Hilfe von Aktivitäten-Diagrammen können Prozessabläufe modelliert und mit Hilfe von Klassendiagrammen analog zu den Aktivitäten-Diagrammen Testdaten spezifiziert werden. Der Nachteil hier ist tatsächlich, dass dies keine BPMN-Modellierung erlaubt, sondern „nur" UML. Eine Integration sowohl für Testmanagementwerkzeuge als auch für Testautomatisierungswerkzeuge ist gegeben.

Die Conformiq Tool Suite (vormals als Conformiq Qtronic bekannt) ist ein modellbasierter Testgenerator kommerzieller Art für funktionale Tests. Die Modellierung kann textuell oder auch über interne oder externe Editoren graphisch durchgeführt werden. Das Modell basiert dabei auf QML (engl.: Qtronic Modeling Language), einer zugeschnittenen Variante der UML. Die Conformiq Tool Suite generiert automatisiert anhand einer Zustandsraumanalyse die entsprechenden Testpläne, Testfälle und ausführbaren Testskripte, je nach Wahl in TTCN-3[71] oder TCL[72] [28].

Auch existiert eine Anbindung sowohl zu gängigen Testmanagementwerkzeugen als auch zu Testautomatisierungswerkzeugen. So können Testfälle direkt in das Testmanagementwerkzeug zur weiteren Verarbeitung transferiert werden. Allerdings müssen diese Testfälle mit zusätzlichem manuellen Aufwand zu automatisierten Testskripten verändert werden, das wäre jedoch nicht so schwerwiegend. Ein weiterer Nachteil ist, dass Anwender eine völlig neue Modellierungsnotation lernen müssen, die viel mehr technisches Know-how voraussetzt, dies ist schwerwiegender. Auch erlaubt es RUDI nicht, die Testdaten, die zwischen den verschiedenen Systemen transferiert werden müssen, zu spezifizieren.

Dagegen bietet sich der „Innovator for Business Analyst" in Kombination mit der MBTSuite von der Firma SeppMed als die MBT-Lösung an, welche die Anforderungen dieser Dissertationen zwar nicht im vollen Umfang, aber dennoch am ehesten von allen MBT-Testwerkzeugen erfüllen. Deswegen werden zur Realisierung des in dieser Dissertation vorzustellenden Ansatzes sowohl IBA als auch die MBTSuite verwendet. IBA ist ein Modellierungstool der MID GmbH und die MBTSuite gehört zur SeppMed GmbH. In Kombination mit gängigen Testmanagementwerkzeugen bilden sie die Basis für das modellbasierte End-to-End-Testen verteilter BIS. Sie erfüllen die zentralen Anforderungen, wie die volle BPMN-Unterstützung bei der Modellierung von Testfällen und die einfache Bedienbarkeit dieses Tools. Wobei IBA für das Modellieren von Testfällen „missbraucht" werden soll, da mit diesem Testwerkzeug bisher nur Geschäftsprozesse modelliert werden. Im Rahmen dieser Dissertation soll der Innovator allerdings als Testprozess-Modellierungswerkzeug ausprobiert werden.

[71] Engl.: Testing and Test Control Notation.
[72] Engl.: Testing Control Language.

3.4.5 Integration von Testfallgeneratoren – Modellierungswerkzeuge und Testmanagementtool

Eine Basis, um generell Testfälle zu erstellen, zu verwalten und auch auszuführen, ist mit Testmanagementwerkzeugen gegeben. Testautomatisierungswerkzeuge bieten zusätzlich eine Ausführungseinheit (VBA[73]), um programmierte Testskripte (Testfälle) automatisiert ausführen zu können. Ein großer Vorteil der in VBA programmierten Funktionsbausteine ist, dass sie von gängigen Testautomatisierungswerkzeugen unterstützt werden. Was MBTSuite und IBA konkret leisten müssen, zeigt folgende Aufstellung:

a) Es muss eine konkrete Testanforderung in einem Requirement-Modul entweder im Testmanagementwerkzeug oder in einem anderen dafür vorgesehenen Werkzeug vorliegen. Dadurch lassen sich Funktionseinheiten identifizieren und gleichzeitig spezifizieren.
b) Die Summe der Funktionseinheiten bildet die funktionale Spezifikation, welche für den Testfall als Basis für die Modellierung dient.
c) Testfälle zu den einzelnen am Prozess beteiligten Softwaresystemen müssen spezifiziert werden. So genügt es, die Prozesse zunächst einzeln zu modellieren, um sie dann später zusammenfügen zu können, oder auch nicht.
d) IBA hat zwar den Vorteil, dass es BPMN vollständig unterstützt, allerdings wird für die Modellierung von Geschäftsprozessen eine Einschränkung[74] auf die wichtigsten Elemente vorgenommen [141]. Dazu muss BPMN um ein Testprofil erweitert werden. Das hat wiederum den Vorteil, dass die Anwender nicht mit der Mächtigkeit von BPMN-Elementen „erschlagen“ werden und sich nur die Elemente aneignen sollen, welche für den täglichen Gebrauch vonnöten sind.
e) Um Testfälle zu generieren, nutzt IBA den Testfallgenerator MBTSuite von Sepp-Med. Auch diese Schnittstelle kann eine Fehlerquelle darstellen, die später in der Evaluierungsphase erneut herangezogen werden soll. Sind die Testfälle modelliert, erfolgt mit der MBTSuite die Generierung mittels der in IBA modellierten Modelle und die Transferierung in ein Testmanagementwerkzeug.
f) Eine zentrale Herausforderung besteht darin, mit den generierten Testfällen tatsächlich auch ausführbare Testskripte zu adaptieren.

3.4.6 Aus welchen Komponenten setzt sich der Lösungsansatz zusammen?

Das Ziel ist es, eine Bibliothek von Problemlösungen für End-to-End-Tests für verteilte BIS zu entwickeln. Mit Hilfe von Modellierungen und Bibliotheken sollen Redundanzen vermieden und Synergien besser genutzt werden. Durch die Modellierung soll nicht nur der Test beschleunigt und verbessert werden, auch sollen Tester schneller auf Änderungen im System reagieren können.

---

[73] Visual Basic.
[74] http://www.bpmn-tool.com/tutorial/.

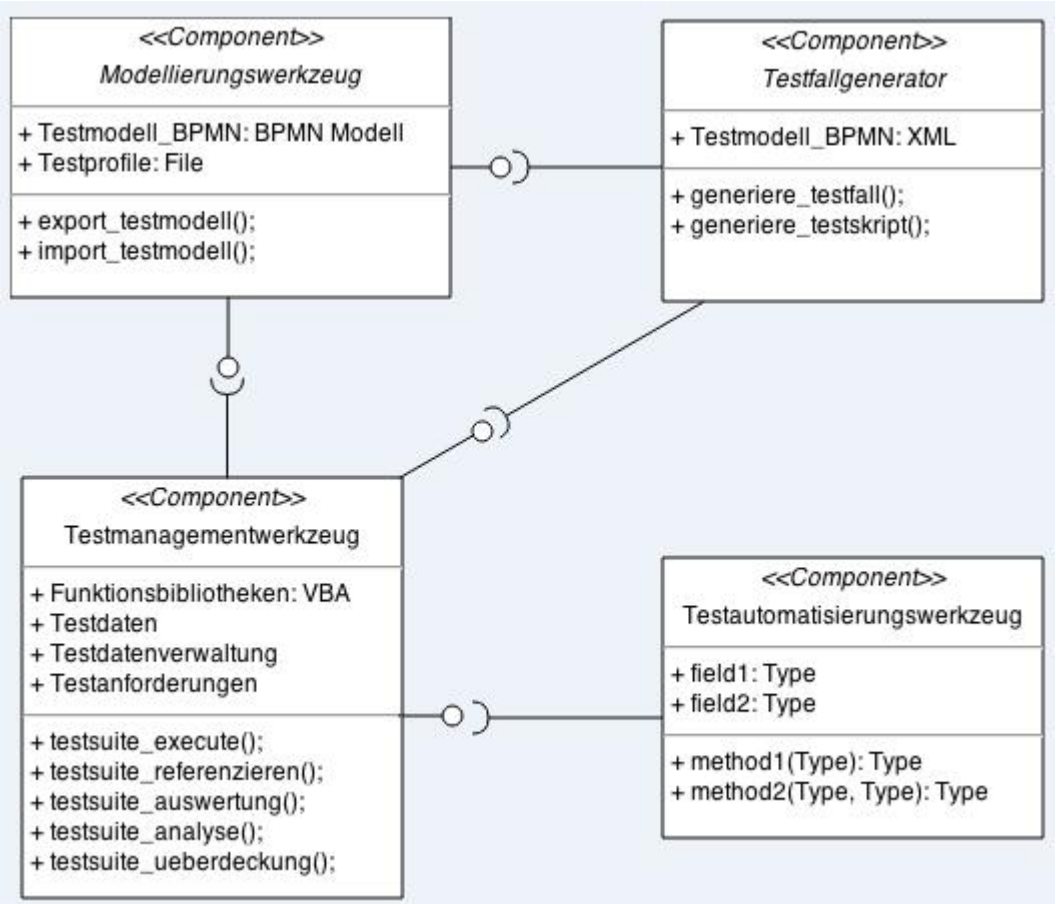


Abbildung 26: Am Lösungsszenario beteiligte Komponenten

Bei der Parametrisierung der Testmodelle mit Testdaten wird folgendes Vorgehen bevorzugt: Alle Aspekte, die zu einer Veränderung des Softwareverhaltens führen, sollten auch mit Testdaten parametrisiert werden. Alle anderen Parameter, die für den Test nicht als relevant klassifiziert werden, sollten nicht ins Testmodell übernommen werden. Der Grund hierfür ist, dass erstens die Komplexität aus den Modellen genommen werden soll (nicht zu verwechseln mit der Komplexität der Applikation) und zweitens nicht unnötig viele Testfälle generiert werden sollen, die evtl. auch nicht genutzt werden und das Testen nur „verkomplizieren“. Testdaten werden mit Hilfe von Klassendiagrammen erfasst und an BPMN-Modelle geheftet.

Ein Testmanagement agiert als Schaltzentrale, dort müssen alle für den Test notwendigen Testdaten, Funktionsbibliotheken und Testdokumentationen abgelegt sein. Sowohl für ein Modellierungswerkzeug als auch für einen Testfallgenerator bietet ein Testmanagementtool einen Ablageort für die generierten Testfälle und die erstellten Modelle. Die Verantwortung für die Testfallverwaltung obliegt dem Testmanagementtool. Mit einem im Testmanagementtool integrierten Requirement-Modul können jetzt die ersten Testanforderungen spezifiziert werden, welche in konkrete Testfälle münden, so dass jede Anforderung einem Testfall zugeordnet werden kann.

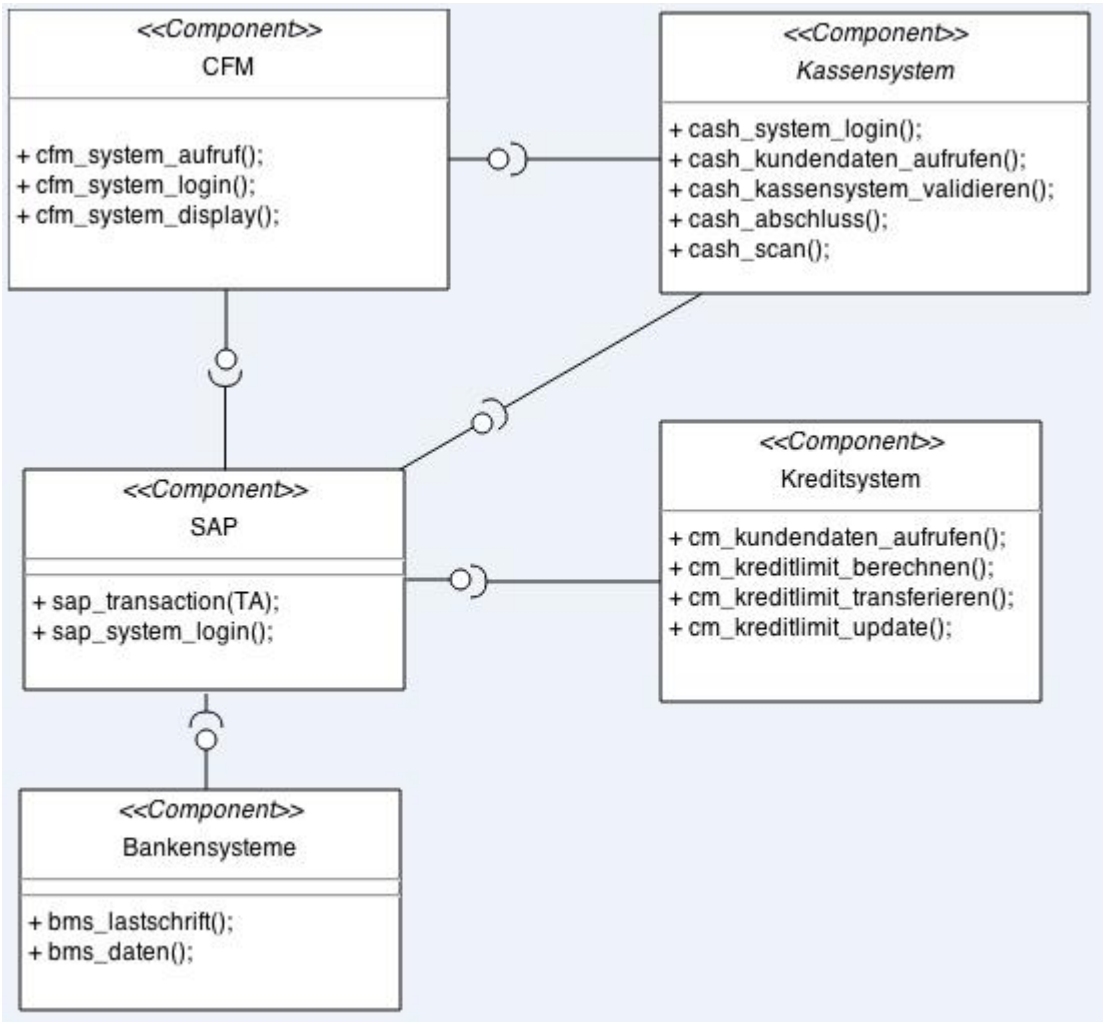


Abbildung 27: Am Testprozess beteiligte Softwaresysteme

3.5 Erweiterung der BPMN-Spezifikation und des Metamodels um Testelemente

Dieser Abschnitt beschreibt kurz, welche Erweiterungen konkret im Rahmen dieser Dissertation sowohl an den Testwerkzeugen als auch an den Modellierungswerkzeugen gemacht werden müssen, unabhängig von der konkreten Implementierung einzelner Programmabschnitte, die zur Umsetzung benötigt werden. Auf die hier kurz skizzierten und zusammengefassten Erweiterungen folgt dann im nächsten Kapitel die detaillierte Erläuterung.

3.5.1 Lösungsansatz Erweiterung 1: IBA[75]-Modellierungswerkzeug, BPMN[76]-Testelemente

Das IBA-Werkzeug unterstützt bei der Modellierung von Geschäftsprozessen. Das Werkzeug kann durch eine Erweiterung auch für die Modellierung von Testprozessen angewendet werden. Dazu wird IBA zusätzlich um Profile erweitert, welche die zu verwendenden

[75] Innovator for Business Analyst.

[76] Business Process Modeling Notation.

Modellelemente definieren. Die Erweiterung definiert neben den grundsätzlichen Eigenschaften, Wertebereichen und Zuordnungen auch die Darstellungsweise, wie die zu verwendenden Symbole. Die Erweiterung enthält zusätzlich Konfigurationen zu Prüfroutinen und Beschreibungsfunktionen. Auch können Bedingungen spezifiziert werden, welche eintreten müssen, bevor eine bestimmte Aktion ausgeführt werden kann. Alle diese Regeln sind jetzt in einer Erweiterung unter dem IBA-Regelwerk abgelegt und als Datei übertragbar, auch auf andere Modellierungswerkzeuge. Ein spezielles Profil für BPMN sorgt für eine Beschränkung von BPMN auf die testrelevanten Elemente.

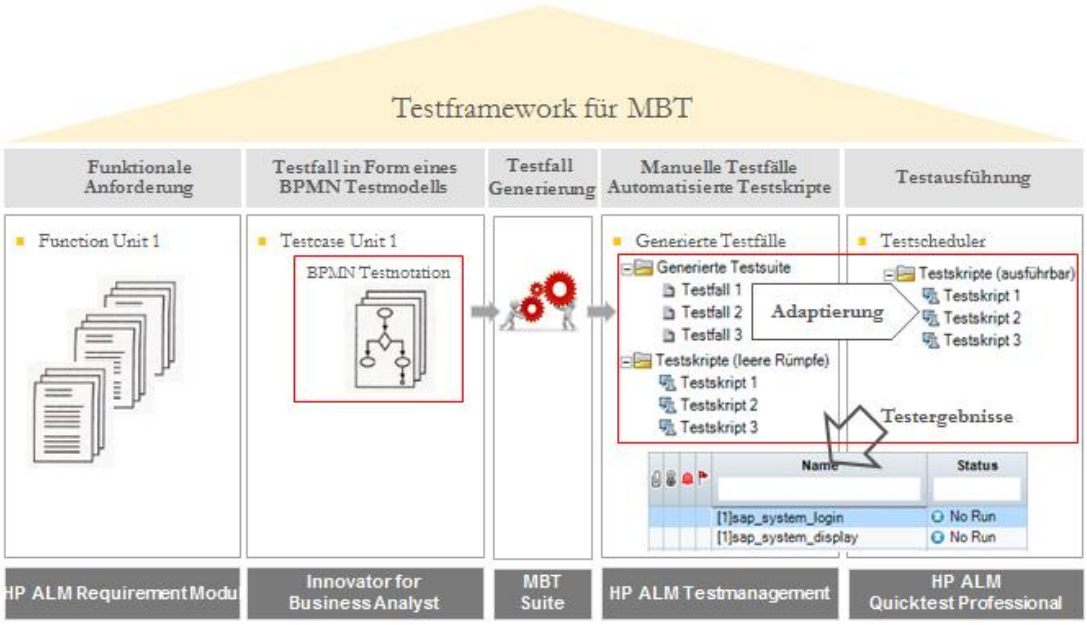


Abbildung 28: Benötigte Erweiterungen

Der Vorteil ist, dass sowohl Tester als auch Mitarbeiter von Fachbereichen diese Notation schnell erlernen können. Mit Hilfe der in der Erweiterung spezifizierten Regeln erfolgt auch eine syntaktische Prüfung der erstellten Testmodelle auf Korrektheit. Die rot umrandeten Elemente sind die Erweiterungen, welche konkret durchgeführt werden.

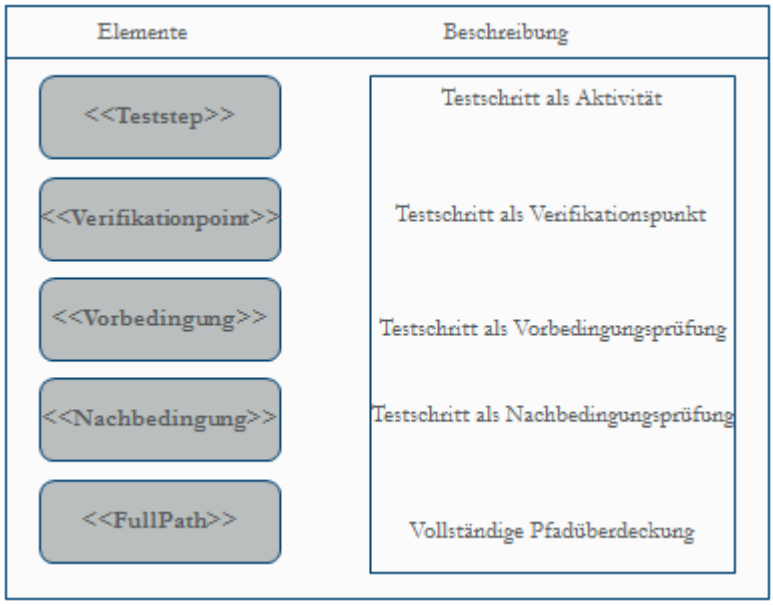


Abbildung 29: Zum BPMN-Standard hinzugefügte Elemente sind rot umrandet

Die durchgeführte Erweiterung führt dazu, dass ein Profil jetzt als eine eigenständige, abgeschlossene Einheit erscheint. Das Konstrukt einer solchen Erweiterung ist das Stereotyp, welches einen Teil eines Profils definiert und auf existierende Metaklassen angewendet wird. Die Erweiterung der BPMN-Elemente um Testspezifika sowie die Erweiterung der Testwerkzeuge ermöglichen jetzt eine Erprobung der Methode. Einen kleinen Ausschnitt, mit welchen Elementen die Spezifikation von BPMN [163] erweitert wird, bietet obige Abbildung.

Für die Dissertation werden zwei Kategorien von Elementen benötigt, zunächst einmal die Testelemente, welche als Aktivitäten definiert sind, und Verbindungsobjekte (Klasse TestElement), die gleichzeitig für Bedingungen und Einschränkungen genutzt werden können.

Das Element (Teststep) repräsentiert immer eine einzelne Aktivität/Aktion im Prozess. Das Symbol wird der in BPMN bereits existierenden Kategorie (Activities) [164] zugeordnet. Das Element (VP) repräsentiert Prüfungspunkte in einem Modell und wird genauso der Kategorie (Activities) hinzugefügt. Die beiden Elemente (Pre, Post) müssen auf zwei Kategorien erweitert werden. Zunächst als eine Aktivität in der Kategorie (Activities), dann in der Kategorie der Verbindungsobjekte. Die Aktivität (FPC) dient dazu, sicherzustellen, dass jeder Pfad eines Modells vollständig durchlaufen wird. Die Aktivität (NP) sorgt dafür, dass einzelne Testschritte in einem Modell losgelöst generiert werden können, ohne den vollständigen Testpfad zu generieren.

Neben diesen Testelementen, ausgedrückt in BPMN-Aktivitäten, können zusätzlich Loop-Wiederholungen spezifiziert werden. Die Vor- und Nachbedingungen können sowohl als eigene Aktivitäten modelliert als auch direkt im zu modellierenden Pfad definiert werden. Diese Erweiterung macht es jetzt einem Testfallgenerator möglich, die Testpfade zu identifizieren und je nach Generierungsstrategie die Testfälle nacheinander aus den Testmodellen abzuleiten. Gleichzeitig muss mit der Erweiterung der Elemente eine Erweiterung der Beschreibung innerhalb von IBA für diese Testelemente erfolgen. Genau diese Beschreibun-

gen greift sich der Testfallgenerator. Folgende Parameter müssen zusätzlich bei der Testschrittmodellierung spezifiziert werden:

Parameter
Testfallname
Testfallbeschreibung
Priorität
Anzahl der Testdurchläufe (min)
Anzahl der Testdurchläufe (max)
Test mit Wiederholungsschleife

Tabelle 1: Testparameter im Testmodell

Diese zusätzlichen Felder, welche aufgrund der Erweiterungen in BPMN und resultierend in IBA nötig werden, können jetzt so konfiguriert werden, dass sie immer eine Anzahl an Initialwerten aufweisen. So können bei der Priorisierung bspw. Werte von 1–4 hinterlegt sein. Bei der Anzahl der minimalen und der maximalen Durchläufe können auch Initialwerte hinterlegt sein. Bei einem Testfall, bei dem eine Anmeldung beim Softwaresystem erfolgen soll, kann der maximale Durchlauf auf 3 gesetzt werden, danach sollte die Anmeldung erfolgt sein oder ein anderer Testschritt sollte ausgelöst werden. Mit Hilfe der Prioritäten lässt sich auch gleichzeitig die Anzahl der Testfälle steuern.

3.5.2 Erweiterung 2: IBA-Brücke zur MBTSuite und Testwerkzeug HPQC[77] für die Adaptierung

Die zweite Erweiterung erfolgt direkt in den Testwerkzeugen, welche für die Ausführung und die Verwaltung der Testfälle genutzt werden sollen. Sowohl in der IBA-Konfiguration als auch in der MBTSuite müssen Einstellungen vorgenommen werden, welche in zwei (Jar-)Dateien zusammengefasst werden und zusammen mit Seppmed und mit der MID GmbH erfolgen. Diese beiden Dateien beinhalten sowohl die Interpretationsregeln der MBTSuite als auch die notwendigen Parametereinstellungen auf der IBA-Seite. Ab Kapitelabschnitt 3.6 wird genauer erläutert, in welche Verzeichnisse diese Dateien kopiert werden müssen.

Nachdem die Konfigurationen abgeschlossen sind, ist die Brücke zwischen IBA und der MBTSuite vorhanden und es können Testmodelle aus der MBTSuite nicht nur importiert/exportiert werden, sondern sie können aufgrund der realisierten Erweiterung von BPMN mit Testelementen interpretiert werden. Anhand der Testnotationen erkennt die MBTSuite die Aktivitäten als Testelemente und generiert daraus Testfälle.

[77] Hewlett Packard Quality Center (Marktführer (60% Marktanteil) als Testmanagementwerkzeug), Quelle: Worldwide Distributed Automated Software Quality Tools 2007–2011.

Konkret müssen aus den in BPMN generierten Testfällen, welche einen manuellen „Charakter“ haben, fertig automatisierte Testskripte gemacht werden. Das ist deswegen notwendig, weil sowohl IBA als auch die MBTSuite nur in der Lage sind, manuelle Testfälle oder Testskripte ohne Implementierungslogik zu generieren. An dieser Stelle setzt jetzt die Adaptierung ein, welche die generierten Testskripte direkt auf implementierte Funktionsbibliotheken referenziert. Damit der Testfall automatisiert ausgeführt werden kann, muss die jeweils passende Funktion in der Funktionsbibliothek mit einer Testablauflogik versehen sein. Die Adaptierung wird konkret an einem Beispiel in Kapitel 3.7 demonstriert.

3.5.3 Erweiterung 3: Testwerkzeug HPQC und der Zugriff auf den abstrakten Syntaxbaum von QTP

Eine weitere Erweiterung leitet sich aus dem Ansatz ab, eine Adaptierung umzusetzen. Hier muss für die Adaptierung ein Zugriff auf den abstrakten Syntaxbaum von QTP ermöglicht werden. Dadurch kann ein Programm analysieren, ob zum generierten Testskript (noch ohne Implementierungslogik) eine Funktion in der Funktionsbibliothek (Identifizierung anhand von zuvor definierten Namenskonventionen) schon vorhanden ist oder nicht. Auch sorgt dieses Programmstück dafür, dass zu jedem generierten Testskript, falls zuvor keine implementierte Funktion in der Funktionsbibliothek vorhanden sein sollte, eine Meldung ausgegeben wird, die den verantwortlichen (Modellierer, Tester, Fachbereich) auffordert, zu diesem generierten Testskript eine Funktion zu definieren.

3.5.4 Erweiterung 4: Generische Funktionsbibliotheken für ERP-Softwaresysteme definieren

Die Anlage der Funktionsbibliotheken als generische Funktionen für standardisierte Softwaresysteme wie die ERP-Systeme sorgt dafür, dass alle einmal erstellten Funktionen und aufgezeichneten Komponenten in der QTP Repository wiederverwendbar werden.

3.5.5 Einschränkungen – Bedingungen bei der Umsetzung der BPMN-Erweiterung

Die Notation und das Metamodell der BPMN-Spezifikation sind unter bestimmten Bedingungen erweiterbar. Es gibt sowohl allgemeine als auch formale Bedingungen. An eine Erweiterung der Notation werden allgemeine Bedingungen gestellt. An eine Erweiterung des Metamodels können sowohl allgemeine als auch formale Bedingungen gestellt werden [83]. Die Erweiterung der Notation durch neue Symbole darf bspw. das „Look and Feel“ [163] der bereits existierenden Symbole von BPMN [83] nicht beeinträchtigen. Unabhängig von der modellierenden Person sollte ein erweitertes Prozessmodell weiterhin leicht verständlich sein. Für die Testnotation setzt der Autor dieser Dissertation daher auf schon bestehende Symbole für die Modellierung. Was das Metamodell angeht, dürfen existierende Elemente durch zusätzliche Attribute erweitert werden [164]. Macht die Abbildung eines speziellen fachlichen Ablaufs das Hinzufügen neuer Elemente erforderlich, dürfen diese die Ausführungssemantik existierender Elemente nicht verletzen [83], [44]. Damit bleibt der

BPMN-Core valide. Abhängig von der Implementierung des verwendeten Modellierungswerkzeugs, können bei einem Austausch die neuen Elemente (samt Attributen) verloren gehen. Die Erprobung erfolgt hier am Modellierungswerkzeug IBA. Eine Übertragbarkeit auf andere Modellierungswerkzeuge wird im Rahmen dieser Dissertation weder erprobt noch evaluiert.

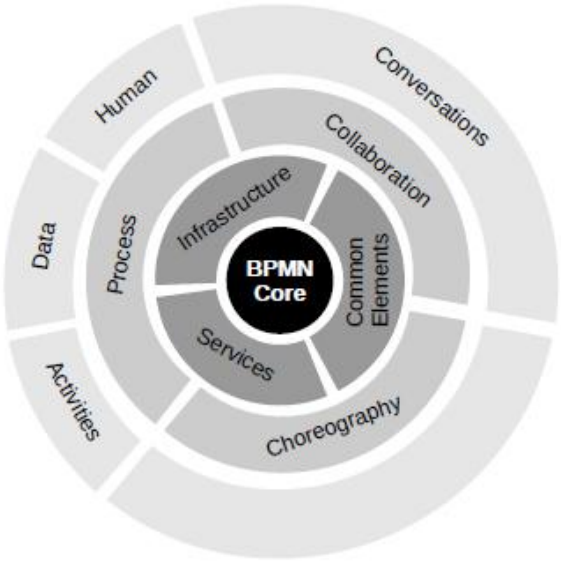


Abbildung 30: BPMN-Core, angelehnt an [163], [164]

Der BPMN-Core bildet die Basis der BPMN, er ist robust, präzise und enthält alle Modellierungen von Prozessen und von notwendigen Konstrukten [83]. Der Core ist in drei Teile gegliedert: „Foundation“ (enthält die grundlegenden Konstrukte der BPMN und der Erweiterbarkeit), „Common“ (für die den Prozessen gemeinsamen Konstrukte) und „Services“ (Konstrukte von Schnittstellen und Diensten).

Ein weiteres BPMN-Core-Paket bildet das „Infrastructure-Paket“, dieses enthält die Klassendefinitionen und die Importe. Die Implementierung dieser Klassen ist für die Modellierung in BPMN sowie für die Realisierung der Austauschbarkeit unerlässlich. Sie legen die konkrete und die abstrakte Syntax als XML-Schema-Definition (XSD) fest. Durch die von der OMG definierten Schnittstellen kann mit IBA das Metamodell eingelesen und mit Hilfe des Konfigurators schrittweise auf die Testnotation erweitert werden. Damit dies reibungslos funktioniert, hat die OMG sich darauf konzentriert, die Trennung zwischen Metamodell und Notation immer beizubehalten [44], [163], [164]. Das Extension-Class-Diagramm ist das UML-Klassendiagramm [164] des Metamodels einer Erweiterung.

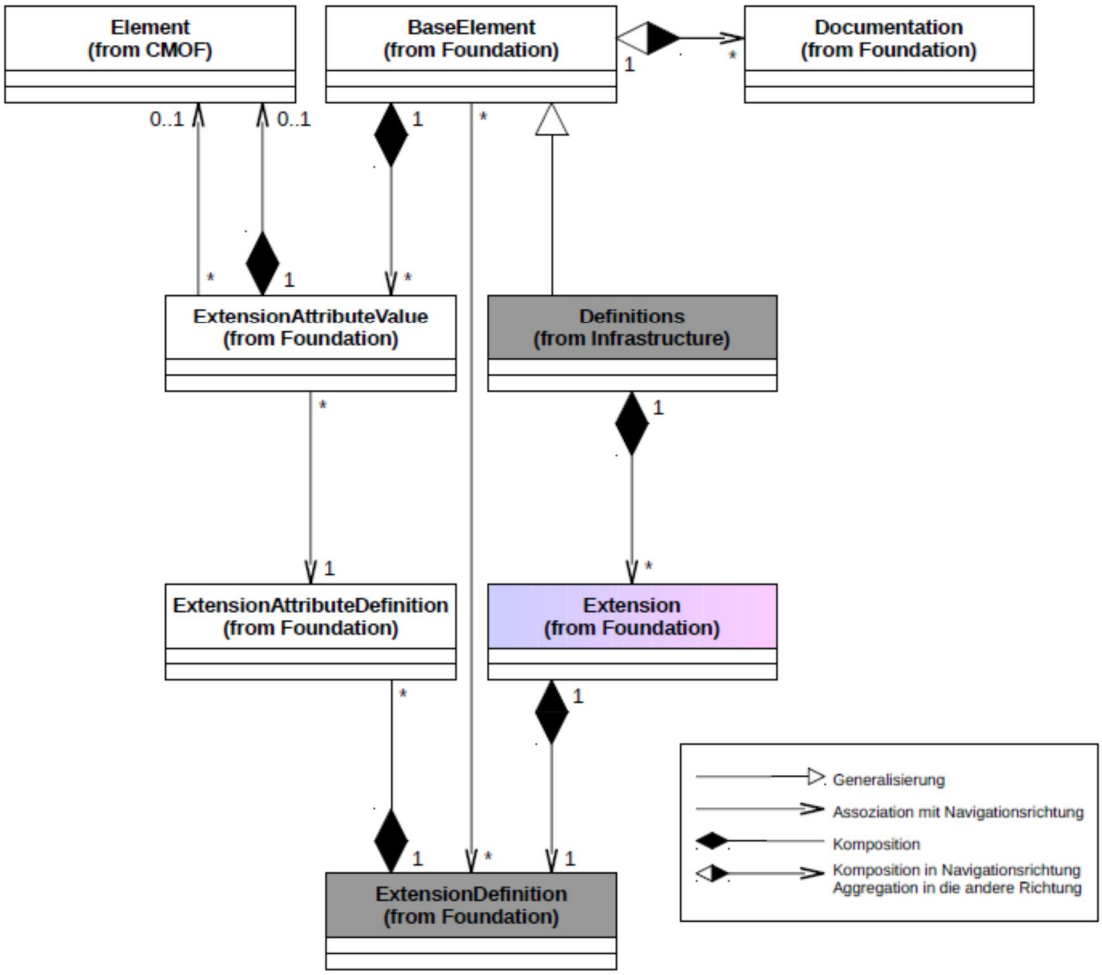


Abbildung 31: BPMN-Erweiterung: Metamodel, nach [163]

„Definitions“, „ExtensionDefinition“, „Extension“ und „ExtensionAttributeDefinition“ sind die für eine Erweiterung der BPMN wesentlichen Klassen. Diese erlauben es, existierende Elemente durch zusätzliche Attribute und durch das Metamodell der BPMN zu erweitern.

Das Metamodell befindet sich im IBA-Installationsverzeichnis im Systemordner „inoprj“. Zusätzliche Attribute existierender Elemente werden einer Instanz der Klasse „Definition“ aus dem Paket „Infrastructure“ hinzugefügt. Um das Metamodell der BPMN durch ein zusätzliches Metamodell zu erweitern, müssen Instanzen der Klassen „Extension“ und „ExtensionAttributeDefinition“ jetzt als Spezialisierung der Klasse „ExtensionDefinition“ aus dem Paket „Foundation“ konzipiert [163], [164] werden.

3.5.6 Erweiterung des Metamodels für die Testnotation – Verbindungsobjekte

Die Erweiterung erfolgt in der Programmiersprache BPEL, welche zwar in XML geschrieben, aber dennoch eine Programmiersprache ist. Die Erweiterungen im Modell werden in XML definiert.

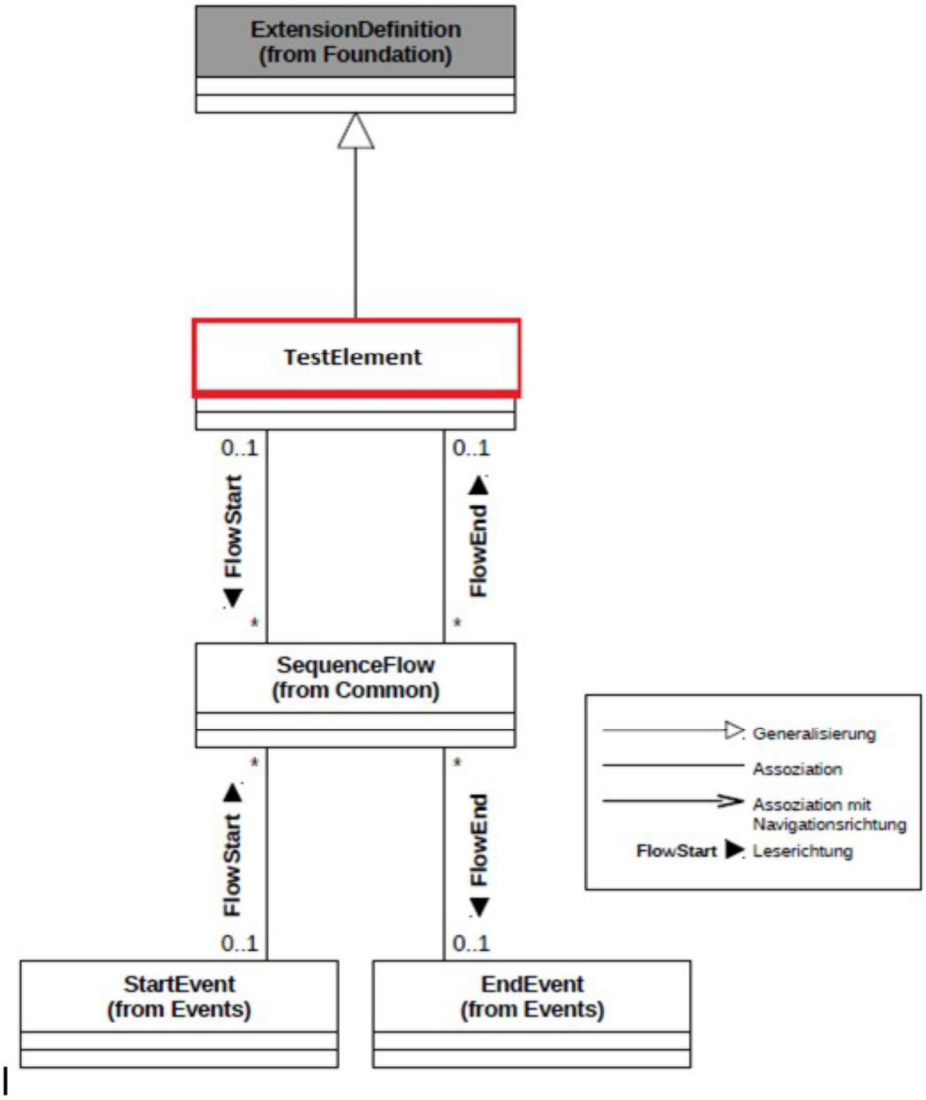


Abbildung 32: Metamodel, erweitert durch den Autor dieser Dissertation um Testelemente und Verbindungsobjekte

In diesem kurzen Abschnitt wird die statische Struktur des hier erweiterten Metamodels als UML-Klassendiagramm dargestellt. Das zusätzliche Metamodell erfüllt alle formalen Bedingungen nach [163] und [164]. Das zusätzliche Metamodell ist als Spezialisierung der Klasse „ExtensionDefinition“ aus dem Paket „Foundation“ [164] konzipiert. Es ist in die Kategorien „Verbindungsobjekte“ und „Aktivitäten“ unterteilt. Eine Instanz der Klasse „SequenceFlow“ (Verbindungsobjekte) geht hierbei die Assoziationen „FlowStart“ und „FlowEnd“ ein. „FlowStart“ und „FlowEnd“ können mit einer oder keiner Instanz der Klasse „TestElement“ assoziiert sein. Die rot umrandete Klasse ist die Klasse, welche hier im Metamodell hinzugefügt wurde und von der „ExtensionDefinition“-Klasse alle Eigenschaften erbt. In dieser Klasse sind alle zu erweiternden Verbindungsobjekte und Aktivitäten, welche in der obigen Grafik dargestellt sind, zusammenfassend enthalten. Als Beispiel für Assoziationen zu existierenden Elementen der BPMN sind die Klassen „StartEvent“ und „EndEvent“ aus dem Paket [163] Events dargestellt. „FlowStart“ kann mit einer oder keiner Instanz der Klasse „StartEvent“ assoziiert sein. „FlowEnd“ ist mit der Instanz der Klasse „EndEvent“ assoziiert. Folgender kleiner Ausschnitt illustriert, wie ein Testschritt in Form einer Aktivität in der BPMN-Spezifikation erweitert wurde:

```
{
  "type":"node",
  "id":"Teststep",
  "title":"BPMN_Teststep",
  "groups":["Activities"],
  "view":"../../extensions/bpmnTestelement_B2B/view/activity/Teststep.svg",
  "icon":"../../extensions/bpmnTestelement_B2B/icons/activity/Teststep.png",
  "properties":[/*...*/]
  }
```

Schritt 2 definiert das Testelement (Aktivität) als Testschritt mit dem Typen „node“ (Knoten), welcher die ID „Teststep“ bekommt. Als Titel kann jetzt „BPMN Teststep“ definiert werden. Es empfiehlt sich, eine feste Namenskonvention einzuhalten, um Überschreibungen zu vermeiden. Dieses neu definierte Element wird jetzt in der BPMN-Kategorie „Activities“ hinzugefügt [83], [164]. Die Vor- und Nachbedingungen sowie die Verifikationspunkte werden sowohl als Knoten typisiert als auch als Kanten. Die Werte von „view“ und „icon“ verweisen auf die Realisierung des Symbols der Zeichenfläche und auf die Bereitstellung des Symbols neben der Zeichenfläche. Es empfiehlt sich, den BPMN-Standard anzuwenden [164]. Analog dazu müssen die Erweiterungen für alle weiteren Knoten und Kanten durchgeführt werden. Bei den Verbindungsobjekten kann auch auf das BPMN-Regelwerk zurückgegriffen werden, wobei Bedingungen, welche direkt im Modell definiert werden müssen, zusätzlich zu einem Regelwerk hinzugefügt werden müssen.

```
"roles":[
"sequence_start",
"sequence_end",
"TeststepFlow_start",
"TeststepFlow_end",
"PartsFlow_start",
"PartsFlow_end"
]
```

Die Definition der „roles“ (Rollen) der BPMN-Testnotation spiegelt die direkten Beziehungen aller Elemente der Erweiterung wider. Die vollständig spezifizierten Erweiterungen der BPMN und des Metamodels befinden sich unter der Systemdatei IBA - c:\innovator\11.5\„innovator.aob“.

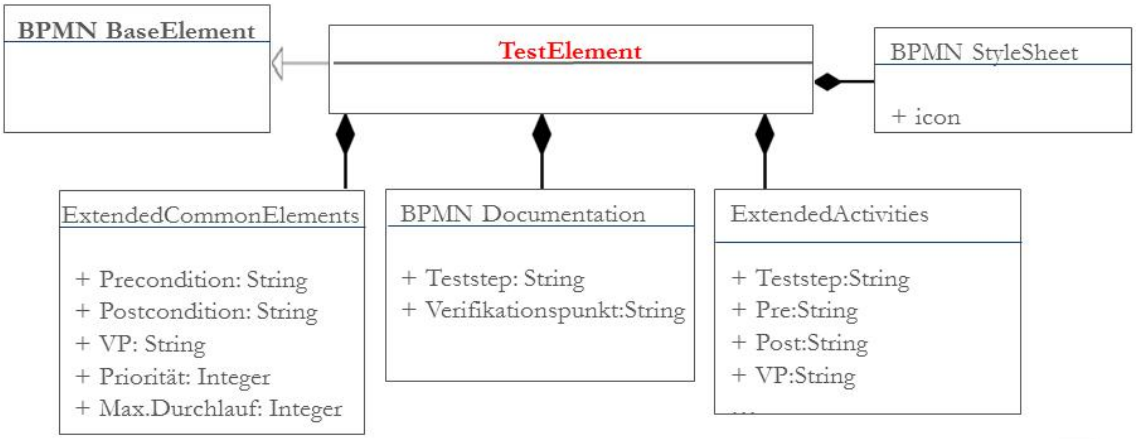
```
"connectionRules":[ {
"role":"preconditionFlow",
"connects":[ {
"from":"preconditions_start",
"to":["preconditions_end"]
} ] }, {
"role":"PartsFlow",
"connects":[ {
"from":"PartsFlow_start",
"to":["PartsFlow_end"]
} ] } ]
```

Wird einem Testschritt eine Bedingung zugeordnet oder wird er spezifiziert, dann kann er indirekte Beziehungen zu all den Testschritten eingehen, denen „roles“ und „rules“ zugeordnet sind [83].

Die Umsetzung der hier vorgestellten Erweiterung erfolgt im Konfigurationseditor des Modellierungswerkzeugs IBA [121]. Für die Umsetzung des Metamodels und der Beziehungen und Bedingungsverknüpfungen zwischen den Elementen muss die Datei „innova-

tor.aob“ im Installationsverzeichnis Innovator dementsprechend erweitert werden. Nach der Erweiterung wird die Datei vom Autor dieser Dissertation in „B2B.aob“ umbenannt, damit kann dieses Profil inklusive aller Erweiterungen als Testnotation genutzt werden.

Diese Datei wird dann im Systemordner „inoprj“ gespeichert. Zusätzlich muss im Ordner „inodir“ ein zusätzlicher Ordner („seppmed“) angelegt werden.



Die Dateien werden zur Registrierung durch den Testfallgenerator und durch das Modellierungswerkzeug ausgelesen. Inwiefern sich die Spezifikationserweiterung auch auf andere Modellierungswerkzeuge anwenden lässt, ist nicht Gegenstand dieser Dissertation und muss explizit untersucht werden. Die vollständige BPMN-Spezifikation befindet sich im Testprofil und kann im Anhang[78] betrachtet werden.

3.5.7 Einführung von Namenskonventionen

Der Testfallgenerator hat in erster Linie dafür zu sorgen, die mit IBA modellierten Testfälle nach einer auszuwählenden Generierungsstrategie[79] im Testfallgenerator (MBTSuite) textuell ins HPQC zu übertragen. Die MBTSuite ist jetzt in der Lage, durch die Erweiterung der BPMN-Spezifikation die Testelemente zu interpretieren und nach der Spezifikation zu generieren. Im HPQC werden die Testfälle in einer zuvor festgelegten Ablagestruktur abgelegt, versioniert und verwaltet. Wichtig und grundlegend für das Gelingen dieses im Rahmen der Dissertation vorzustellenden Lösungsansatzes sind einerseits die Einführung einer allgemeingültigen Namenskonvention für Testfälle/Testskripte und, korrespondierend damit, die Einführung der Funktionsbibliotheken. Die für diese Dissertation zu verwendenden Namenskonventionen [126], [131] für die Testfälle sind wie folgt aufgebaut:

<<Softwaresystem(Abkürzung)_Funktionsbaustein_Testfallname>>

Als Beispiel soll ein SAP-Testfall dienen, welcher das Anmelden bei einem SAP-System beschreibt. So sind das Softwaresystem „SAP“ und der Baustein „Login“ zu nennen. Hinter einem Funktionsbaustein können sich dann mehrere einzelne Testschritte verbergen, z. B. alle notwendigen Schritte, die für ein erfolgreiches Anmelden bei einem SAP-System erforderlich sind. Als Testfallname kann spezifiziert werden, welche Aktionen dieser Test-

[78] Alle Dateien sind in einem Datenträger zusammengefasst und werden dieser Dissertation beigelegt.
[79] Pfadüberdeckung (vollständige und nicht vollständige Pfadüberdeckung, Zweigüberdeckung, Kantenüberdeckung, Anweisungsüberdeckung) – Prozessüberdeckung (vom Autor dieser Dissertation bevorzugte Strategie, welche während der Ausarbeitung dieses Kapitels noch erläutert werden soll).

fall konkret ausführen soll, wie z. B. die Anmeldung (Eingeben von Benutzername, Passwort, Systemname usw.) im SAP-System. Zusätzlich muss eine Ordnerstruktur angelegt werden, um Testfälle einheitlich und strukturiert abzulegen.

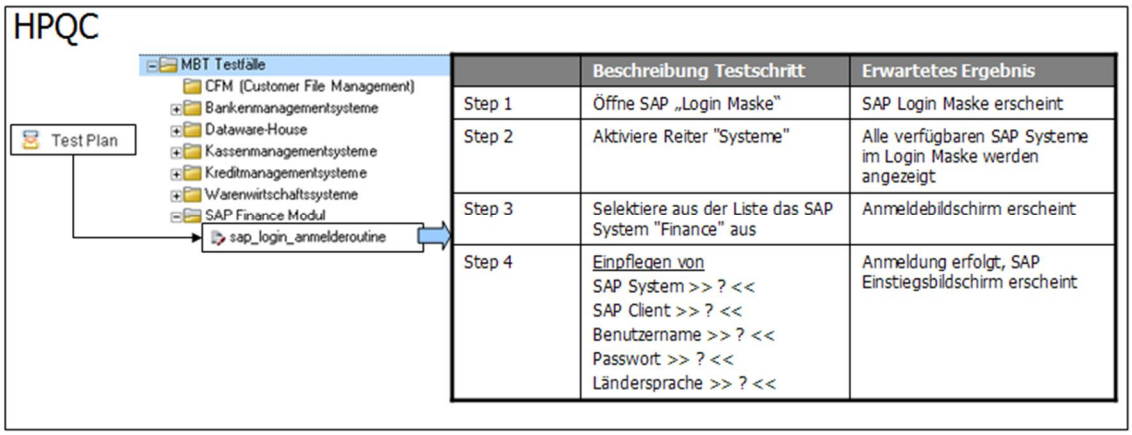


Abbildung 33: HPQC-Namenskonventionen und Ablagestruktur

Einen entscheidenden Beitrag leisten die festgelegten Namenskonventionen bei der Testautomatisierung. Angesichts der immer noch bei MBT-Testwerkzeugen herrschenden Probleme, aus generierten manuellen Testfällen tatsächlich auch automatisierte und fertig ausführbare Testskripte abzuleiten [2], [31], [116], können Namenskonventionen dazu dienen, eine Lösung zu unterstützen. Generell muss in jeder Unternehmensorganisation, welche ein Testmanagement besitzt, über einheitliche Namenskonventionen nachgedacht werden [116]. Diese dienen auch dem einheitlichen Sprachgebrauch unter Entwicklern, Testern und Fachbereichen.

Aber auch eine festgelegte Ablagestruktur trägt zu einer Strukturierung der Tests bei, so müssen die generierten Testfälle und Testskripte in vordefinierten Ordnern abgelegt werden, damit bspw. Testskripte mit Hilfe von Funktionsbibliotheken auf diese Testfälle zugreifen können. So können in „Testskripte" Referenzen auf Testfälle hinterlegt werden oder diese können wiederum auf Testskripte und Testdaten verweisen.

3.6 Erweiterung mit BPMN und die Einbindung von Testwerkzeugen

Damit mit der Modellierung begonnen werden kann, müssen technische Konfigurationen durchgeführt werden, welche das Zusammenspiel zwischen den einzelnen Testwerkzeugen im Verbund mit dem Modellierungswerkzeug erst erlauben. Sowohl HPQC als auch QTP können für diese Dissertation als Vollversionen eingesetzt werden. Dank der Kooperation mit der MID GmbH kann gleichzeitig das Modellierungswerkzeug IBA für die Evaluierung und die Demonstration des Fallbeispiels im Rahmen dieser Dissertation als Vollversion (Enterprise Version) genutzt werden. Auch wurden die für die Dissertation notwendigen Konfigurationen an IBA-Systemdateien, welche für die Anbindung an die MBTSuite essentiell sind, Dank der Zustimmung der Partner durch den Autor dieser Dissertation vorgenommen.

Da IBA kein spezielles Testmodellierungswerkzeug darstellt, muss ein spezielles Testprofil angelegt werden, welches mit Hilfe eines „Importers" das Testmodell von IBA in die

MBTSuite überführt. Mit Hilfe der Erweiterungen in der BPMN-Spezifikation kann jetzt ein Testprofil angefertigt werden, mit dem genau die benötigten Testelemente für die Testfallmodellierung angewendet werden können. Auch dienen das anzulegende Testprofil und die Konfiguration der Einhaltung von Modellierungsregeln sowie der Bereitstellung von Spezifizierungen wie Vorbedingungen, Nachbedingungen, Schleifen und Verifikationspunkte (Prüfpunkte) im Modell.

3.6.1 BPMN-Testelemente in die Modellstruktur von BPMN und IBA einbinden

Die Modellvorlage „Object eXcellence“, welche bei IBA [121] Bestandteil jeder Modellvorlage ist, kann jetzt genutzt werden, um ein zusätzliches Profil anzulegen. Abbildung 32 demonstrierte bereits die UML-Klassenstruktur. Die Umsetzung, welche mit Hilfe des Editors und des Konfigurators mittels IBA angefertigt wird, wird mit Hilfe des Testprofils als eine Add-in Vorlage gespeichert und wird somit von verschiedenen Testprojekten verwendet. Das „Object eXcellence“-Modell ist das spezielle Systemmodell von IBA und kann wieder 1:N Modelle enthalten. So ist das zu erweiternde Profil ein Teil des Systemmodells, gleichzeitig aber bietet es eine abgeschlossene Sicht auf das System.

Die MBTSuite muss dafür an IBA angebunden sein. Auch das Testmanagementwerkzeug, welches die generierten Testfälle verwaltet und ablegt, muss an die MBTSuite angebunden werden. HPQC nutzt dafür die offenen Schnittstellen, diese erlauben es, die MBTSuite einfach an ein Testmanagementwerkzeug anzubinden. Durch Namenskonventionen (in den Testfällen) werden die Testfälle generiert und in das vorgesehene Verzeichnis im HPQC übertragen.

3.6.2 HPQC-QTP-Einbindung als Voraussetzung

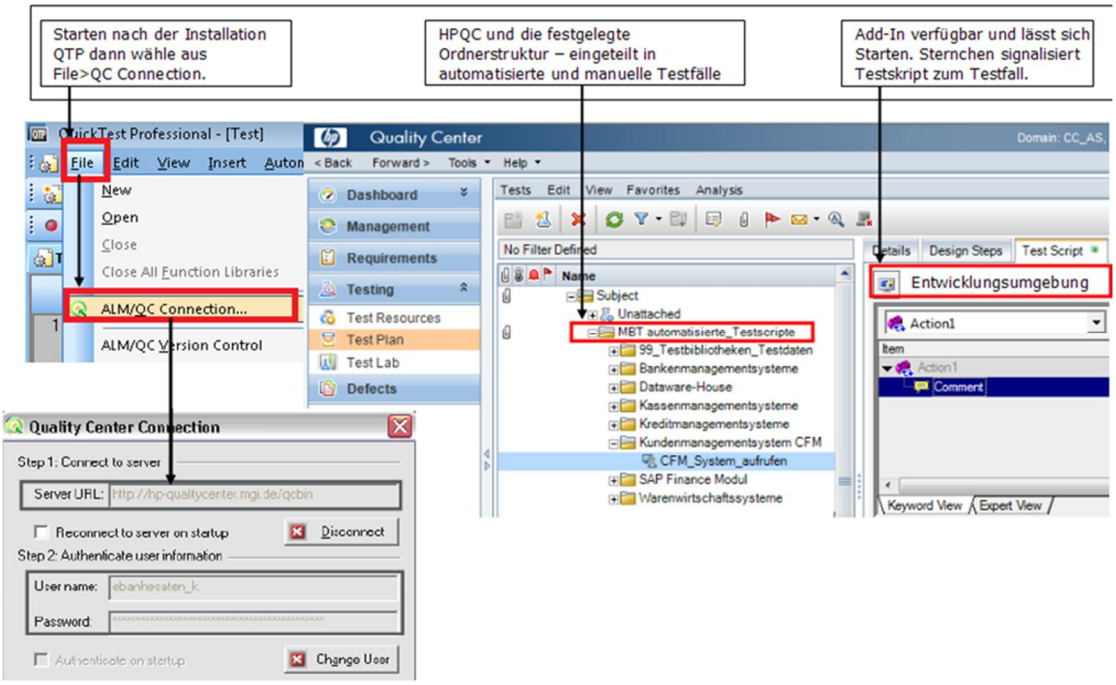


Abbildung 34: Testmanagement und Testautomatisierungswerkzeug

Eine vollständige Installationsanleitung von HPQC und QTP bieten [139], [161]. Verbunden werden die beiden Testwerkzeuge durch das HP-Add-in, welches sich im Standardpaket dieser HP-Produkte befindet. QTP benötigt dafür die Adresse des Applikationsservers des Testmanagementwerkzeuges HPQC. Im HPQC können je nach Test eigene Testprojekte angelegt werden. So muss beim Verbindungsaufbau das jeweilige korrekte Testprojekt mit parametrisiert werden. Durch das Klicken des „connect“-Buttons kann die Verbindung hergestellt werden. Somit können sich jetzt Testskripte aus dem HPQC direkt in QTP laden und umgekehrt [161]. Testskripte benötigen für ihre Ausführung korrespondierende Funktionsbibliotheken mit Testlogik. Diese Testlogiken beinhalten den Geschäftsprozessablauf in programmierten Funktionsbausteinen.

3.6.3 Erweiterung der IBA-Konfiguration als Testprofil und MBTSuite-Brücke

Die Modellierung von Testfällen in BPMN erfordert einen gesonderten Weg, da, wie schon erläutert, IBA nicht speziell die Testfallmodellierung fokussiert. Für die Aufgabenstellung dieser Dissertation ist es daher notwendig, zwei zusätzliche Dateien[80] anzufertigen: eine Profil- und eine Konfigurationsdatei. Diese beiden Dateien basieren auf der BPMN-Spezifikationserweiterung und bilden gemeinsam die Brücke zwischen IBA als Modellierungswerkzeug und der MBTSuite als Testfallgenerator. Diese beiden Dateien erlauben es, Testspezifikationen in Testmodellen auf der Basis von BPMN abzubilden.

Die Konfigurationsdatei ist die Datei, welche das Testprofil beinhaltet, so sind in dieser Datei Konfigurationen hinterlegt, wie Elemente und Stereotypen im Testmodell genau zu

[80] Die SeppMed GmbH in Person von Hr. Dr. Beisser unterstützte den Autor dieser Dissertation bei der Konfiguration und der Definition der Profildatei.

spezifizieren sind. Die Aufgabe des Konfigurators ist es, das Metamodell zu definieren, welches die Grundlage der eigentlichen Modellierung bilden soll. In der Konfigurationsdatei werden die Artefakte und die Organisation eines Modells definiert. Auch werden die Abhängigkeiten und die Übergänge zwischen den einzelnen BPMN-Elementen festgelegt. Diese Festlegungen erfolgen im Rahmen von Profilen in der schon erwähnten Profildatei (B2B.aob).

Die BPMN beschreibt weder eine Methode noch setzt sie Softwareentwicklungsprozesse voraus. Das Metamodell von BPMN definiert lediglich die Semantik und die Notationen der Elemente, die bei der Modellierung genutzt werden. Durch die Definitionen von Stereotypen, Stereotypeneigenschaften sowie Bedingungen und Einschränkungen in Profilen (Testprofil) kann das BPMN-Metamodell somit um diese Testelemente erweitert werden. In der Profildatei werden testspezifische Stereotypen definiert, welche eine spezifische Terminologie umsetzen. Sie definieren die Syntax und die Einschränkungen spezieller Konstrukte, legen Symbole fest, präzisieren die Semantik, bestimmen Artefakte und deren Vorlagen usw.

Diese Erweiterbarkeit der vorhandenen Metamodell-Klassen ist ein Merkmal aller OMG-konformen Metamodelle. So konnte im Rahmen dieser Dissertation das Testprofil „B2B.aob“ angelegt werden, welches speziell für das Testen von Geschäftsprozessen konzipiert ist und Elemente beinhaltet, welche es erlauben, Pakete zu bilden. So ist sichergestellt, dass ein Modell mit dem Stereotyp <<BPMN-Prozessdiagramm>> nur Pakete mit dem Stereotyp <<Prozessdiagramm>> enthalten darf. Analog dazu dürfen Modelle mit dem Stereotyp <<Klassendiagramm>> nur Klassen mit dem Stereotyp <<Klassendiagramm>> zugeordnet werden.

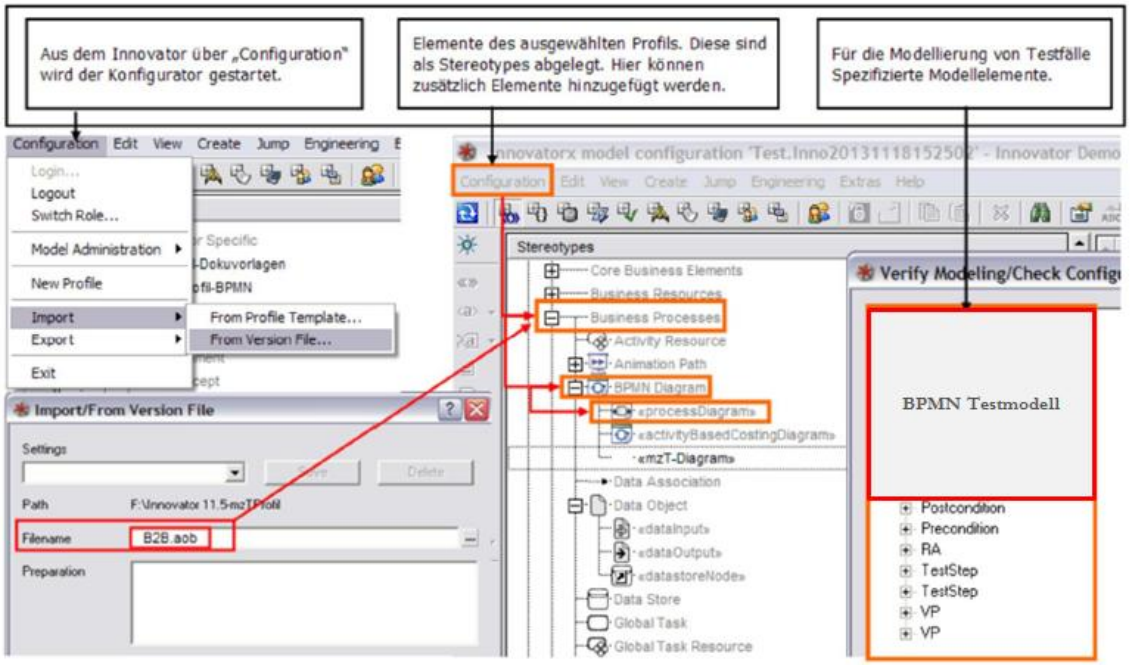


Abbildung 35: Testprofil-Einbindung – Konfiguration

Gleichzeitig ist mit dem hier definierten Testprofil ein einheitlicher Testmodellierungsstil sichergestellt, welcher für einen projektübergreifenden Einsatz von Testfallgeneratoren Voraussetzung ist. IBA bietet einen Konfigurationseditor [85], in dem alle Elemente der Konfiguration eines BPMN-Modells dargestellt und bearbeitet werden können. Wichtig

sind an dieser Stelle die jeweiligen Prüfroutinen, welche festgelegt und bereits in Abbildung 29 gezeigt wurden. Des Weiteren ermöglicht der Konfigurationseditor das Bearbeiten von Profilen und von deren Profilelementen (Stereotype, Stereotypeigenschaften). Auch können Profile importiert bzw. exportiert werden. Die Integration einer Profildatei erfolgt mit Hilfe von IBA und der Importfunktion im Administrationsmodul [85].

Das Erstellen der Profildatei erfolgt direkt im IBA, die dafür benötigten Testelemente sind durch die BPMN-Spezifikationserweiterung vorhanden. Dabei wird ausgenutzt, dass die Elemente sich einfach in einem Explorer darstellen lassen. Abbildung 35 zeigt, aus welchen Bestandteilen die Datei „B2B.aob“ besteht.

In der Datei werden genau die Elemente und Stereotypen erfasst, welche sich durch die MBTSuite unterstützen lassen. Da die MBTSuite ein Testfallgenerator ist, sollen als Empfehlung tatsächlich nur Testelemente damit interpretiert werden. Stereotypen erlauben es jetzt, in BPMN erfasste Testschritte als solche zu spezifizieren. Einen großen Vorteil bietet die Festlegung von Prüfpunkten sowie von Vor- und Nachbedingungen im Modell.

Die Erweiterung der BPMN-Spezifikation macht gleichzeitig eine Spezifikationserweiterung bei IBA notwendig. So müssen zusätzliche Felder wie (TCName, TCDescription, Prio, Duration) hinzugefügt werden, um testspezifische Merkmale auch im Modell erfassen zu können. Dazu wird das Regelwerk der Modellierung, auf welchem IBA basiert, einfach um diese Felder erweitert. Durch die Erweiterung der BPMN-Spezifizierung ist schon der Grundstein gelegt.

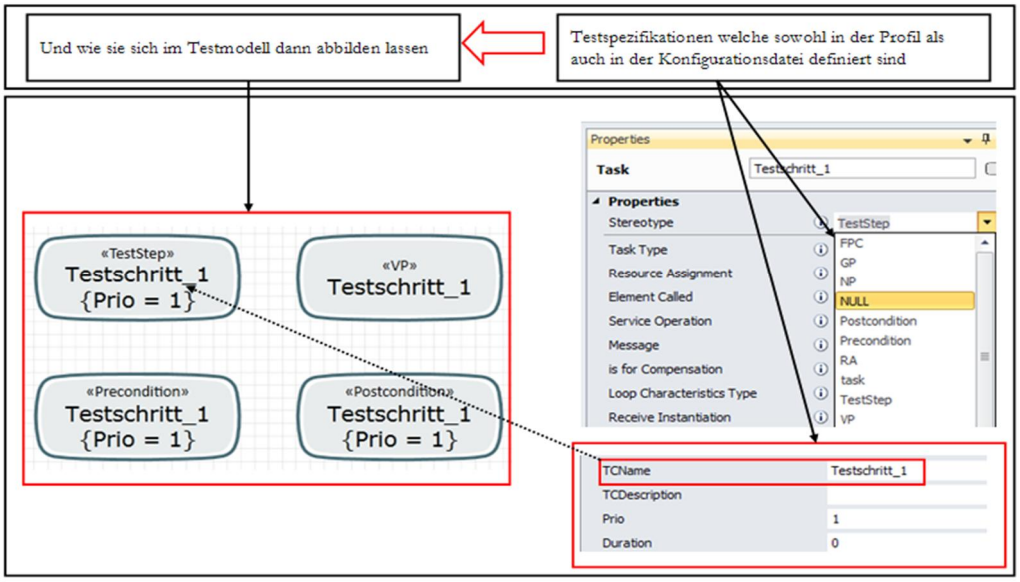


Abbildung 36: Testnotation, Konfiguration Testmodell

Verifikationspunkte sind besonders wichtig, wenn Testdaten überprüft werden müssen. Mit den Verifikationspunkten kann das erwartete Systemverhalten spezifiziert und mit Hilfe eines Testskripts geprüft werden. Durch die Erweiterung lassen sich die mittels IBA zu modellierenden Testschritte/Verifikationspunkte oder auch zu jeder Vor- und Nachbedingung spezielle Testfallnamen und Testfallbeschreibungen hinzufügen. Zusätzlich können Testschritte oder Prüfpunkte direkt mit einer Priorisierung spezifiziert werden. Soll ein

Testschritt mehrmals, aber dennoch mit einer maximalen Anzahl, ausgeführt werden dürfen, so kann dies auch direkt über eine „Loop-Funktion“ im Modell festgelegt werden. Bestes Beispiel ist die Anmelderoutine mit maximal „3 Anmeldefehlversuchen“, bevor der Benutzer gesperrt wird.

Das Testprofil wird in IBA in den Systemordner „inoprj“ kopiert, da dieser für die Validierung benötigt wird. Dabei repräsentiert der Ordner eine Repository, sprich alle erstellten Elemente und Logiken werden im Verbund in einer zentralen Repository in IBA gespeichert. Beim Starten von IBA wird gleichzeitig das Testprofil ausgewählt, welches aus der Repository geladen werden soll.

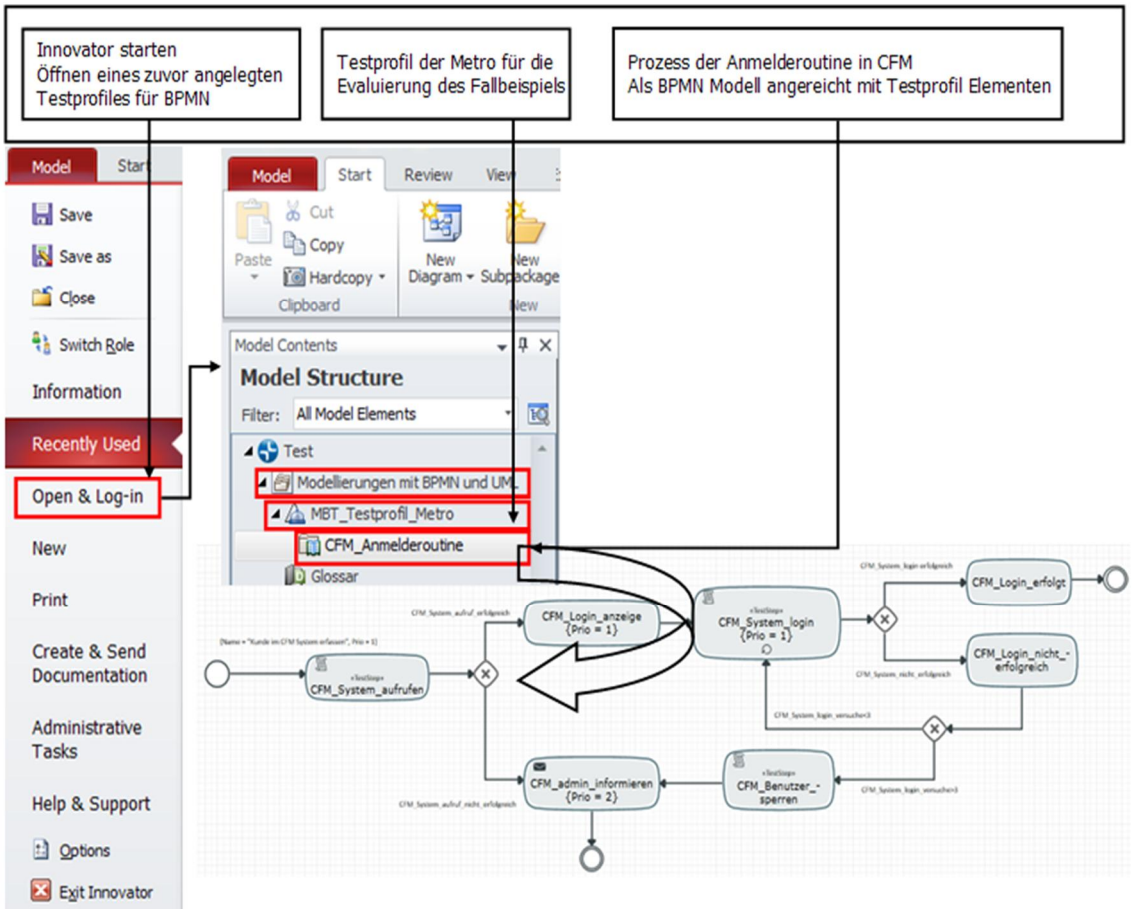


Abbildung 37: Starten des Innovators – Laden des Testprofils und des Testmodells

3.6.3.1 Testfallimport von IBA zur MBTSuite

Die Überleitung eines mittels IBA angefertigten BPMN-Modells kann über zwei Wege erfolgen: Entweder das angefertigte BPMN-Testmodell wird aus IBA mit Hilfe des Befehls „Command“ [121] in die MBTSuite übertragen oder das mittels IBA entworfene und validierte BPMN-Testmodell wird mit Hilfe der Importfunktion in die MBTSuite importiert. Damit der Import bzw. der Export des BPMN-Testmodells fehlerfrei erfolgen kann, muss zwischen diesen beiden agierenden Werkzeugen eine Verbindung aufgebaut werden. Die Verbindung erfolgt über eine Standardschnittstelle, welche speziell für die Testfallmodellierung auf der Basis der in der MBTSuite unterstützten Elemente angefertigt wird.

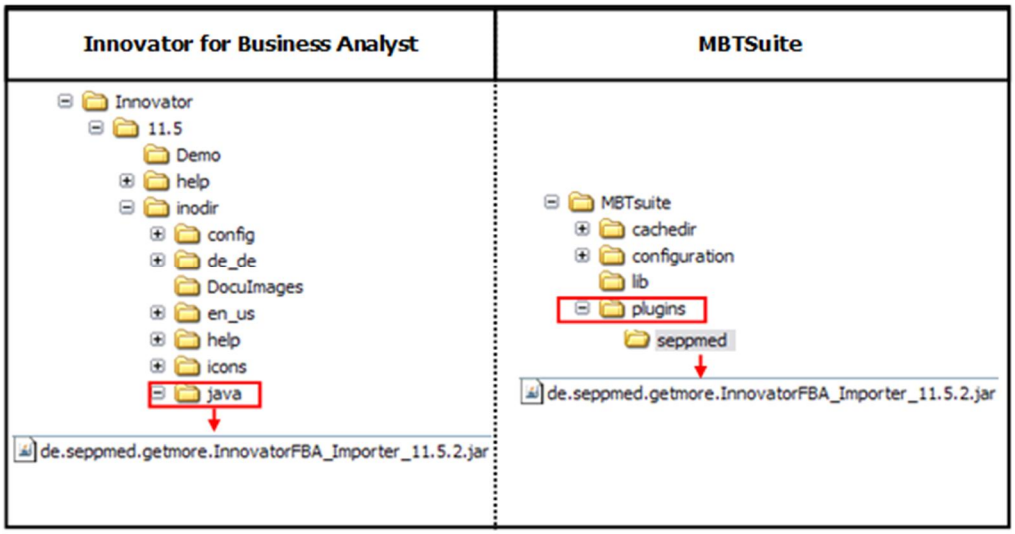


Abbildung 38: Installationsverzeichnisse für die Konfigurationsdateien, welche als Brücke zwischen IBA und MBTSuite agieren

Für die konkrete Verbindung stellt die Seppmed GmbH eine Datei („de.seppmed.getmore.InnovatorFBA_Importer_11.5.2.jar“) zur Verfügung, welche die Kopplung zwischen diesen beiden Werkzeugen übernimmt [121]. Dazu werden die beiden „Importerdateien“ direkt in die beiden Installationsverzeichnisse der Werkzeuge kopiert (siehe obige Abbildung). In der MBTSuite muss dazu ein spezieller Ordner (seppmed) im Unterordner „Plugins“ angelegt werden.

Die Dateien dienen als „Brücke“ zwischen dem Testfallgenerator und dem Modellierungswerkzeug. So ist es möglich, die angefertigten Testmodelle von einem Werkzeug in das andere Werkzeug zu laden, zu modifizieren und auszuführen. Mit Hilfe des zuvor in BPMN angelegten Testprofils und der Testelemente-Erweiterung kann jetzt die MBTSuite diese Elemente nach den in der MBTSuite definierten Regeln interpretieren und Testfälle generieren.

In den Testmodellen können Testschritte und Testprüfpunkte (Verifikationspunkte) direkt als solche gekennzeichnet werden. Diese einmalig zu programmierenden Funktionsbausteine, auf welche dann auf die generierten Testskripte referenzieren, können in wiederverwendbaren Funktionsbibliotheken zusammengefasst werden. An die Testmodelle lassen sich gleichzeitig auch Testdaten in Form von Dateien (Excelformat oder UML-Klassendiagramme) heften. Testergebnisse oder Zwischenergebnisse können direkt im Excelformat für den weiteren Testverlauf zwischengespeichert werden.

3.6.3.2 Konkrete Umsetzung – Begriff Prozessüberdeckung

Ein konkretes Beispiel, wie ein modellierter Prozessschritt mit Hilfe der Generierungsstrategie (Pfadabdeckung) und mit dem Testschritt <<CFM_System_aufrufen>> funktioniert, soll jetzt demonstriert werden. Zunächst wird das erstellte Testmodell in die MBTSuite importiert, direkt im Anschluss soll die anzuwendende Generierungsstrategie ausgewählt werden.

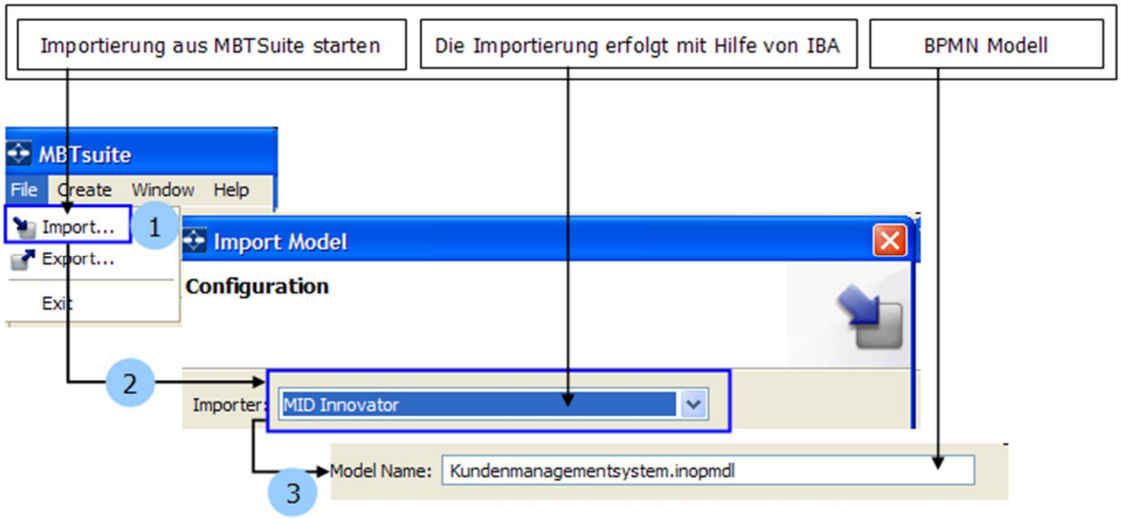


Abbildung 39: Modellimport aus IBA über die MBTSuite

Für die Ausführung des Testmodells können verschiedene Testfallgenerierungsstrategien angewandt werden [121], [30]. Gewählt werden können:

- Kantenüberdeckungsstrategie
- Knotenüberdeckungsstrategie
- Pfadüberdeckungsstrategie

Bei einer vollständigen Pfadüberdeckungsstrategie werden alle Programmpfade im Modell sowie alle Schleifen durchlaufen und ausgewertet. Jeder Pfad in einem Testmodell repräsentiert einen Testfall.

Um eine Testfallexplosion während der Testfallgenerierung zu vermeiden, muss eine Generierungsstrategie gewählt werden, welche die wesentlichen Geschäftsprozesse aus den Testmodellen generiert. Dazu werden die einzelnen Testpfade im Testmodell mit zusätzlichen Parametern wie Prioritäten und Risikoklassifizierungen nach dem Geschäftsprozess versehen. So wird an dieser Stelle eine zusätzliche Generierungsstrategie durch den Autor dieser Dissertation vorgeschlagen: die der Prozessüberdeckung.

Diese Strategie erfordert eine besondere fachliche Expertise in den zu testenden Geschäftsprozessen. Mitarbeiter aus den Fachbereichen müssen zusammen mit Business-Analysten direkt bei der Modellierung jeden Pfad nach seiner Kritikalität [1] bewerten und priorisieren. Das hat gleichzeitig den Vorteil, dass damit eine Diskussion zwischen den Beteiligten (Fachbereich, Tester, Business-Analyst) angeregt wird. So können Fehler früher entdeckt werden (in der Modellierungsphase), was Folgekosten bei Fehlerbehebungen deutlich reduziert [1].

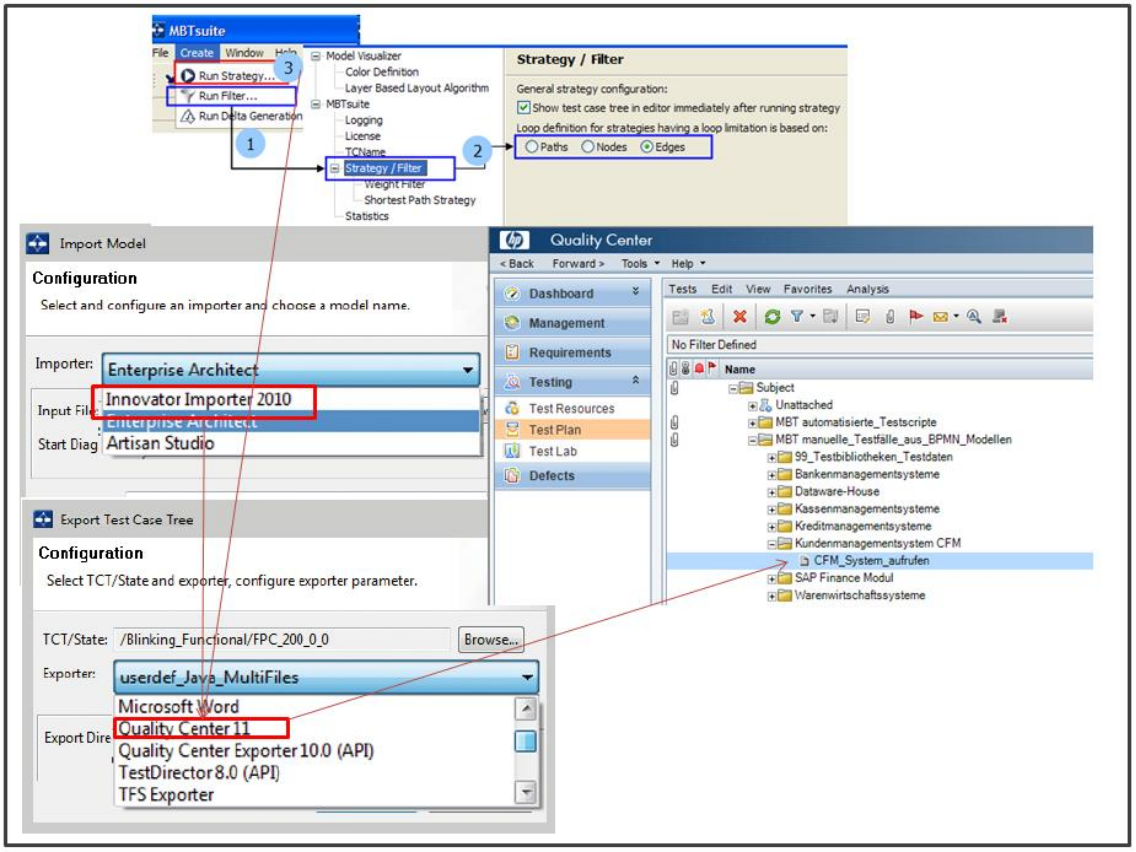


Abbildung 40: Generierungsstrategie und beteiligte Komponenten

Da die Generierungsstrategie der Prozessüberdeckung nicht explizit werkzeugtechnisch unterstützt wird, bedient sich der Autor dieser Dissertation eines kleinen Umwegs, um die Prozessüberdeckung zu realisieren. Die Kanten und Knoten müssen dazu schrittweise und einzeln priorisiert werden. Bei der Generierung der Testfälle werden dann alle Pfade mit einer Priorität von „1“ und „2“ bevorzugt ausgeführt. Pfade, die eine niedrige Priorität besitzen (bspw. „3“ und „4“), können optional ausgeführt werden. Somit ist sichergestellt, dass der Fokus bei der Generierung von Testfällen zunächst einmal auf die geschäftskritischen Prozesse gelegt wird. Nachdem die gewünschte Testgenerierungsstrategie ausgewählt ist, kann im nächsten Schritt (3) mit der Testfallgenerierung aus dem eingelesenen Testmodell begonnen werden.

3.7 Erweiterung und Anfertigung der Adaptierung

Bei der Modellierung der Testfälle kommt es darauf an, die einzelnen Testschritte in BPMN-Aktivitäten (Testschritte) zu modellieren und dadurch gleichzeitig zu strukturieren. Hinter jeder Aktivität steht später ein Testskript, welches die korrespondierenden Testschritte gebündelt in Funktionen (Funktionsbausteine) und wiederum in Funktionsbibliotheken enthält. So besteht bspw. die Aktivität „CFM_System_aufrufen“ aus zusätzlichen Schritten. Dazu gehört es bspw., das CFM-System mit Hilfe eines Betriebssystems (bspw. Windows) auszuwählen, zu starten und zu überprüfen, ob die CFM-Anmeldemaske erscheint. Ob tatsächlich die Anmeldemaske erscheint, kann bspw. durch einen Verifikationspunkt im Testmodell spezifiziert und im Testskript mit einer Prüfroutine überprüft werden. Genau diese einzelnen Testschritte werden so immer als Funktionsbaustein gebündelt einem Testschritt zugeordnet.

Gegenstand dieses Unterkapitels ist es, die einzelnen am Prozess beteiligten Aktivitäten zu modellieren. Dabei beschränkt sich der Autor dieser Dissertation auf die zentralen Aktivitäten im IT-gestützten Zahlungsabwicklungsprozess. Es ist nicht möglich, alle Varianten, mit welchen ein IT-gestützter Zahlungsabwicklungsprozess getestet werden kann, im Rahmen dieser Dissertation darzustellen. Deshalb verweist der Autor dieser Dissertation auf die angefertigten Testmodelle der am Prozess beteiligten Aktivitäten im Anhang dieser Dissertation. Der Fokus im Kernteil dieser Dissertation liegt auf der Modellierung der Kernprozesse bezogen auf das Fallbeispiel und auf deren späterer Bündelung zu Regressionstests.

Neben der Modellierung soll in einem Schritt die Übertragung der Testfälle ins HPQC gezeigt werden. Die Generierung von Testfällen auf der Basis von Modellen ist ein Teil, der zweite Teil besteht darin, aus den generierten Testfällen fertig automatisierte und ausführbare Testskripte anzufertigen. Die Abbildung der generierten Testfälle auf korrespondierende Funktionsbibliotheken wird kurz vorgestellt und es wird gezeigt, welche Methode der Autor dieser Dissertation wählt, um eine 1:1-Zuordnung zwischen Testschritt im Testmodell, generiertem Testfall, Testskript und der zum Testskript passenden Testablauflogik in der Funktionsbibliothek herzustellen.

3.7.1 Testfallmodell Kundenmanagementsystem – Testskript-Adaptierung

Ausgelöst wird ein B2B-Prozess dadurch, dass ein Kunde oder Neukunde im CFM-System angelegt werden muss. Der Prozess wird dabei durch die Anfrage eines Kunden impliziert, diesen als B2B-Kunden anzulegen. Wobei im Rahmen dieser Dissertation immer nur kleine Ausschnitte aus dem BPMN-Testmodell demonstriert werden sollen. Testschritte, welche gleichzeitig Testfälle abbilden, werden bei der Generierung generell nicht von Verifikationspunkten unterschieden.

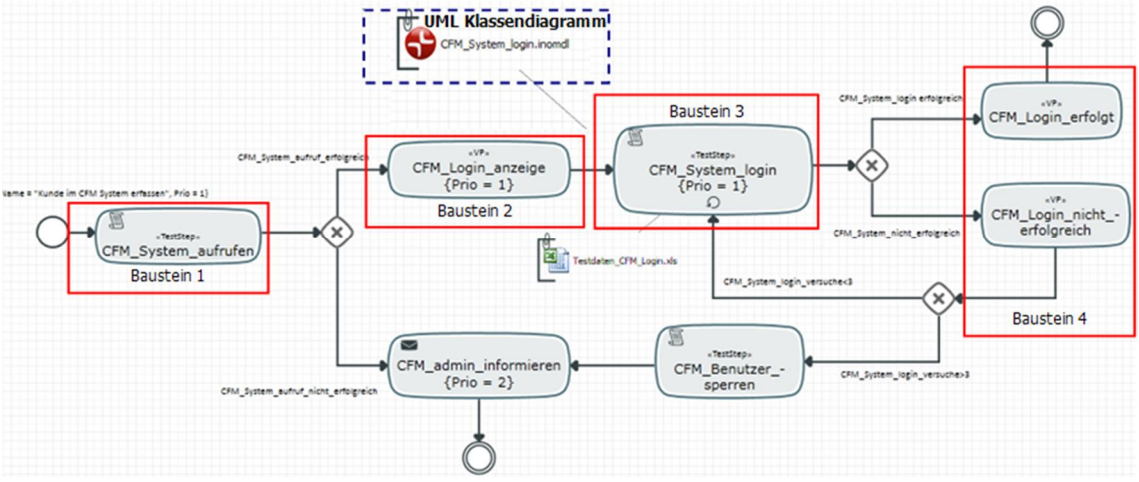


Abbildung 41: BPMN-Testmodell zur Anmelderoutine

Die Voraussetzung, um einen neuen Kunden im Kundenmanagementsystem anzulegen, ist, dass eine berechtigte Person sich mit ihren Benutzerdaten im CFM-System anmeldet. Zusätzlich muss sie im Besitz der passenden Berechtigungen sein, um genau diesen Prozessschritt durchführen zu können.

3.7.1.1 Aktivität „CFM-System aufrufen“ inklusive Adaptierung

Hinter dem Prozess <<CFM_System_aufrufen>> verbergen sich weitere Testschritte. Die einzelnen Testschritte werden mit Hilfe des VBA-Befehls „Call“ aus den einzelnen Testskripten aufgerufen. Diese Testschritte können wiederum in Unterprozesse ausarten, all diese Testprozessschritte werden jeweils in Funktionsbibliotheken eingebunden. Das Ziel ist es, die Testschritte direkt in Funktionsbibliotheken zu kapseln. Der Hintergrund ist, dass der Testfallgenerator nur die jeweiligen im Testmodell erfassten Testschritte generiert. Eine weitere Variante ist, dass der Testfallgenerator ein Testskript aus den im Testmodell hinterlegten Spezifizierungen generiert, allerdings sind die Testfallgeneratoren nur in der Lage, einen Testskriptrumpf in VBA zu generieren. So können immer zwei Arten von Testfällen generiert werden: <<CFM_System_aufrufen>> und analog dazu ein Testskript mit einem Testskriptrumpf mit gleichem Namen und mit dem Unterschied, dass es sich hierbei um ein Testskript handelt.

Beide Testfälle werden ins HPQC übertragen und dort hinsichtlich einer zuvor festgelegten Ablagestruktur abgelegt. Hier können die Testfälle weiter spezifiziert oder direkt einem Testscheduler übergeben werden, welcher dann mit dem Tester zusammen eine Testablaufreihenfolge der Testfälle generieren kann. Bei der Generierung der Testskripte werden jeweils die im Testmodell definierten Aktivitäten als Testschritt definiert [8]. Somit ist nun der erste Grundstein für die Automatisierung dieser Testschritte gelegt.

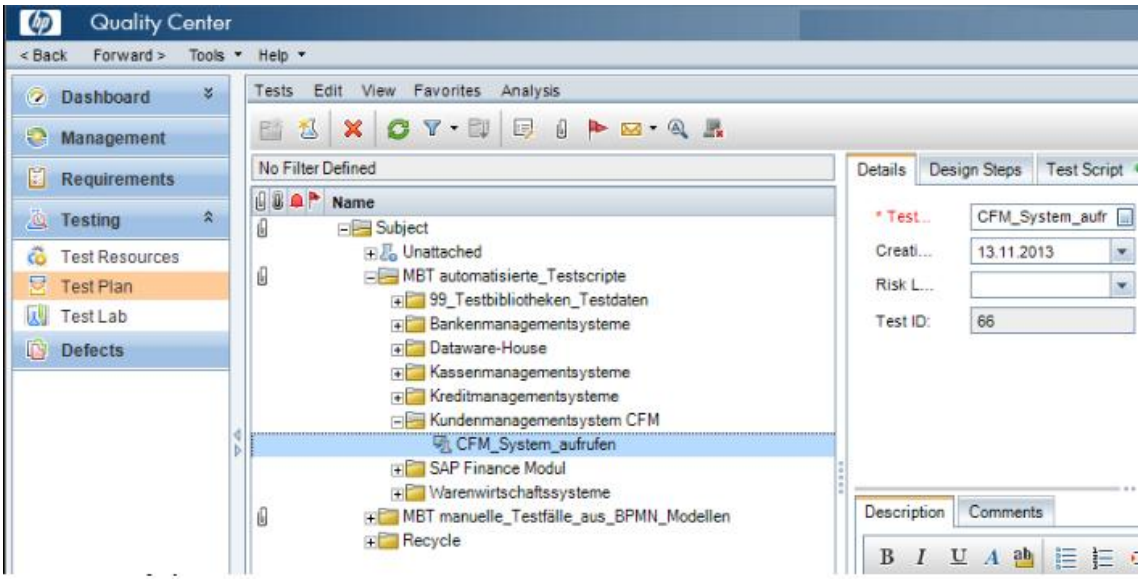


Abbildung 42: Generierung manueller Testfall und Testskriptrumpf

Das Problem, welches sich allerdings hieraus ergibt, ist, dass zwar Testskripte aus einem Testmodell generiert werden können, dass diese allerdings weder Implementierungslogiken noch Referenzen [121] zu bestehenden Funktionsbibliotheken oder Testfällen besitzen.

Für das erste Problem muss in jedem Fall ein einmaliger Initialaufwand aufgebracht werden. So müssen die jeweils hinterlegten Testschritte einmal implementiert und in Funktionsbibliotheken gebündelt und strukturiert werden. Strukturiert deswegen, weil einmal programmierte Funktionalitäten, gebündelt in Bibliotheken, auch von anderen Softwaresystemen zum Testen wiederverwendet werden sollen. Zu jedem Testschritt im Testmodell muss es eine korrespondierende Funktion in der Funktionsbibliothek geben, dieser zu leis-

tende Aufwand bringt viele Vorteile gegenüber dem klassischen Automatisieren von Testfällen. Der zu automatisierende Testschritt ist Bestandteil eines Testmodells. Sind Änderungen an einem Testschritt notwendig, so lassen sich diese einfach am Testmodell direkt erfassen. Gleichzeitig kann auf die extern ausgelagerten Testdaten direkt zugegriffen werden, um bspw. vorhandene Testdaten um weitere Parameter zu erweitern [77]. Als unterstützende Maßnahme bei der späteren Generierung von Testskripten können Geschäftsprozesse im ersten Schritt mit der Aufzeichnungsfunktion (Capture & Replay) mit QTP aufgezeichnet werden.

Das ist die denkbar schlechteste Vorgehensweise bei der Automatisierung von Testprozessen [77], [119], aber genau in diesem Fall unterstützt dies bei der automatischen Codegenerierung als Grundlage für die spätere deskriptive Programmierung. Die deskriptive Programmierung übernimmt dann die Codierung nach Schlüsselwörtern der zu testenden Applikation. Jedes Softwareelement wird in der QTP Repository abgelegt und somit ist ein Testskript in der Lage, auf diese Komponenten mit Hilfe von VBA-Funktionen zuzugreifen. Damit ist das erste Problem zwar mit Aufwand verbunden, aber dennoch mit diesem Ansatz zu meistern.

Das zweite Problem macht hingegen einen Eingriff in das Testmanagementtool notwendig. Das Ziel ist es, generierte Testskripte (Testskriptrumpfe) direkt auf die korrespondierenden Implementierungen in den Funktionsbibliotheken zu referenzieren. Über diesen kleinen „Umweg“ werden aus den im Testmodell definierten Testschritten fertig ausführbare Testskripte generiert. Damit allerdings ein generierter Testskriptrumpf direkt auf eine korrespondierende implementierte Funktionsbibliothek (oder Funktion) zugreifen kann, muss eine zusätzliche Referenz entwickelt werden. Dazu ist es nötig, dass der Autor dieser Dissertation am HPQC-Standard, welcher von HP an die Kunden ausgeliefert wird, eine Erweiterung (Krücke) vornimmt.[81] Ein zusätzliches Feld („Test Name“) muss hinzugefügt und mit einer Funktionslogik in VBA erweitert werden. Die so gebaute „Krücke“ übernimmt die Adaptierungen und erlaubt es, dass die generierten Testskripte direkt auf korrespondierende Funktionen in den jeweiligen Funktionsbibliotheken referenzieren. Durch die zuvor festgelegten Namenskonventionen für Testfälle kann analog dazu die jeweils korrespondierende Funktion in der Funktionsbibliothek benannt werden. Die jeweiligen Funktionen beginnen immer mit dem jeweils zu testenden Softwaresystem, wie CFM, und darauf folgen dann der Testfallname und die durchzuführende Aktion.

Mit Hilfe des QTP-Syntaxbaumes [161], welcher alle implementierten Funktionen in QTP enthält, können die generierten Testskriptrumpfe direkt einer Funktion im Syntaxbaum zugeordnet werden. Die implementierten Funktionen in der Funktionsbibliothek:

```
Public Function GetTestName (objObjRange, strRole, row, strUser, i, strCCCol,
                             appFixedValues)

Public Sub Find_Function_in_Functionlibary

Public Sub Function RefineOutputFile (objApp, AttachmentName)

Sub keepAttInTestRun (strDirName, strFileName)
```

[81] HP Deutschland, Produktverantwortung: Hr. Stephan Mocken. Mit diesem wurde vereinbart, dass der HPQC-Standard um ein Programmstück erweitert werden muss. Da dies eine Modifizierung des Standards darstellt, musste eine Genehmigung durch HP erfolgen.

Sub getAllAttFromTestSetTest (flib, strTempDir)

Function getAttFromTestObject (flib, strAttachName, strTempDir)

Function getAttFromTestSetTest (flib, strTempDir, strFileName)

Function testplan_path ()

sorgen dafür, dass im ersten Schritt der abstrakte Syntaxbaum von QTP mit all den in ihm implementierten Funktionen ausgelesen werden kann. Unter Zuhilfenahme des jetzt zusätzlich implementierten Eingabefeldes im HPQC „Test Name“ kann der Testfall (Testskriptrumpf) ausgewählt werden, den die MBTSuite zuvor aus dem Testmodell generiert hat. Mit dem Auswählen des Testfallnamens wird eine Prüfung ausgelöst, welche sowohl die Funktion „GetTestName“ als auch die Funktion „Find_Function_in_Functionlibary“ implizit ausführt. Das Resultat dieser Prüfung ist, zu ermitteln, ob auf der Basis dieser übergebenen Testfallnamen eine korrespondierende Funktion in der Funktionsbibliothek existiert und eine Testlogik enthält. Ist das der Fall, so sorgt die Unterfunktion „keepAttInTestRun“ dafür, dass im generierten Testskript der Befehl „Call“ auf genau die ermittelte Funktion gesetzt wird. Damit ist eine Referenz hergestellt. Ist der Befehl „Call“ gesetzt, so wird nichts gemacht. Finden die Funktionen keine korrespondierenden Funktionen in der Funktionsbibliothek, so erfolgt die Meldung, dass eine Funktion mit Testlogik zu genau diesem Testskript zu implementieren ist. Sowohl das Suchen im Syntaxbaum als auch das Setzen des Befehls „Call“ im Testskriptrumpf, welcher die Referenzierung übernimmt, geschehen automatisiert.

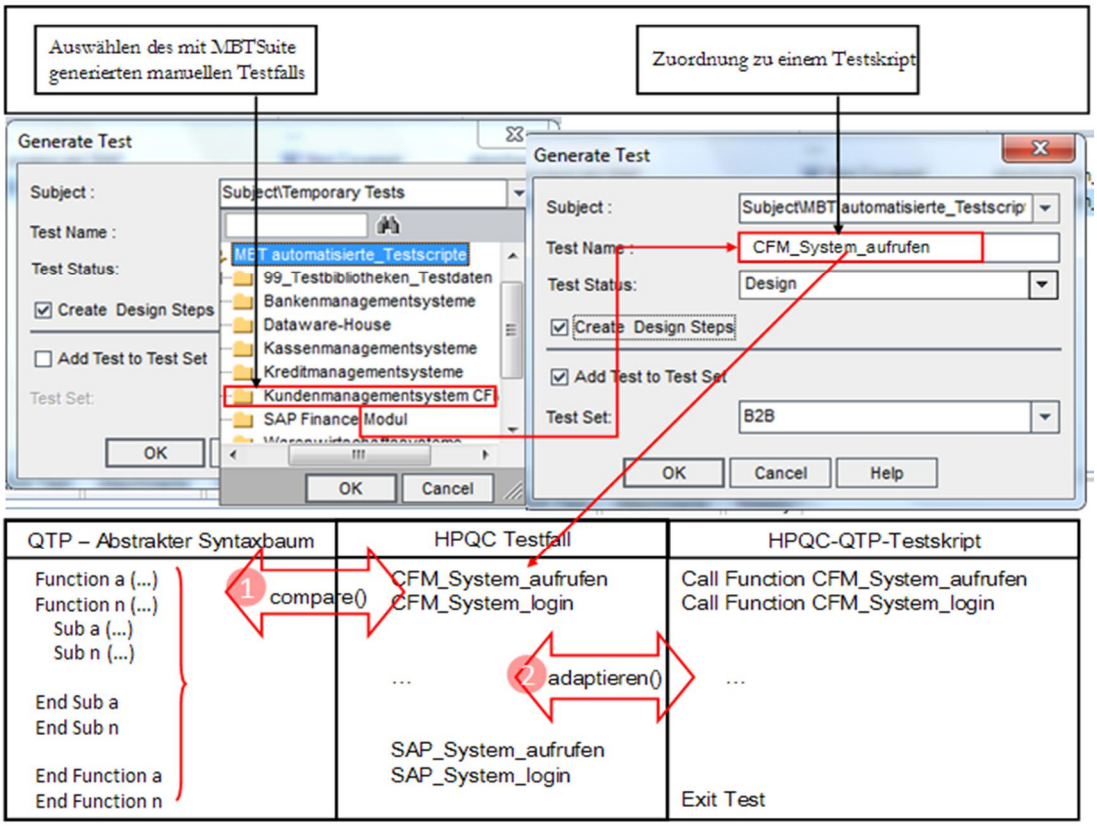


Abbildung 43: Adaptierungsprozess – Zugriff auf AST (abstrakter Syntaxbaum)

Am Ende der Selektionen und Zuordnungen stehen zwei Listen: eine Liste von Testschritten, zu denen jeweils Funktionen in Funktionsbibliotheken implementiert werden müssen.

Und in einer zweiten Liste sind alle Testschritte aufgelistet, zu denen bereits Funktionen in der Funktionsbibliothek vorhanden sind. So ist sichergestellt, dass zu jedem im Testmodell modellierten Testschritt mindestens eine Funktion existiert, welche die jeweils passende Testlogik beinhaltet und automatisiert ausgeführt werden kann. Zwingende Voraussetzung für das Gelingen dieser Methode der Adaptierung ist die Einhaltung der Namenskonventionen bei der Testfallbenennung und bei der korrespondierenden Funktionsbenennung in den Funktionsbibliotheken. Das Ausführen der Testskripte übernimmt ein Testscheduler im HPQC.

Ein Testscheduler [161] ist Bestandteil eines jeden Testmanagementtools und kann dafür genutzt werden, implementierte Funktionen in einer geordneten Reihenfolge und mit einer Zeitscheibe versehen aufzurufen. Die Referenz auf die zum Test passenden Testdaten im HPQC ist dagegen einfach. An dieser Stelle existiert eine generische Funktion, welche entwickelt wird und auf welche dann alle anderen Testskripte immer wieder referenzieren können. Da der Fokus hier auf dem Regressionstest liegt, bringt das Auslagern der Testdaten auch noch einen weiteren Vorteil. So sind alle Ergänzungen und Spezifizierungen, welche an den Testdaten nachträglich vorgenommen werden, nach einem erneuten Generierungsprozess nicht verloren. Testschritte können aufgrund von Änderungen in Modellierungen immer variiert werden, aber die dazugehörigen Testdaten bleiben davon zunächst unberührt. Zur Verdeutlichung soll jetzt anhand des Beispiels <<CFM_System_aufrufen>> demonstriert werden, wie Testskript, Funktion und Funktionsbibliothek während eines Testlaufs miteinander agieren. Der Testfallgenerator generiert den im Modell erfassten Pfad und den dazugehörenden Testfall <<CFM_System_aufrufen>>. Dieser Testfall besteht im Wesentlichen aus zwei Schritten: das Programm CFM aus dem Kontext eines Betriebssystems „auswählen“ und „starten“. Wie die Codierung dazu aussehen kann, zeigt folgender Abschnitt.

3.7.1.2 Funktionsbibliothek CFM-System aufrufen – Testdatenspezifizierung

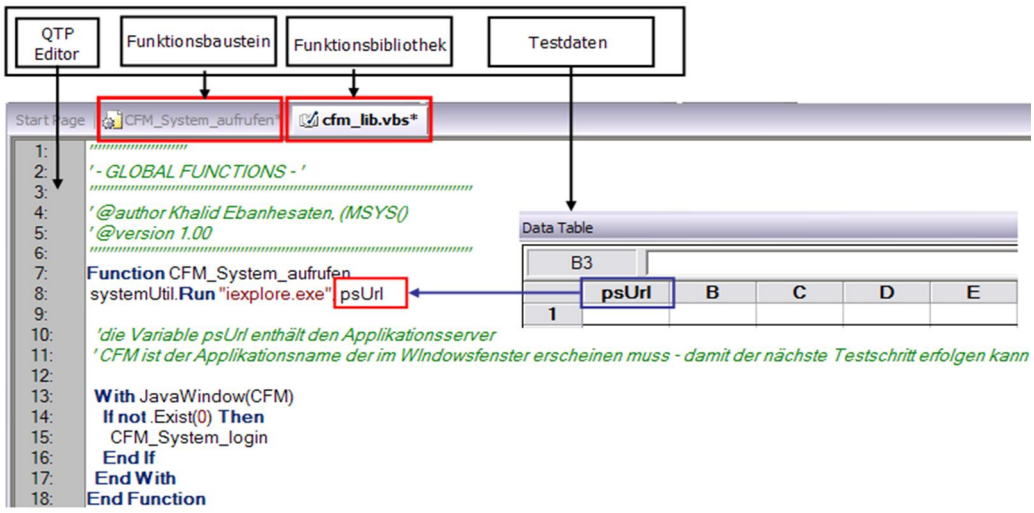


Abbildung 44: Erstellung Funktionsbaustein für die Funktionsbibliothek (Adaptierung)

Das vom Testfallgenerator generierte Testskript mit der leeren Funktion „CFM_System_aufrufen“ muss jetzt mit Testablauflogik implementiert werden. Alle für

das Softwaresystem CFM implementierten Testschritte werden in der Funktionsbibliothek „cfm_lib.vbs“ zusammengefasst. Im betrachteten Beispiel des Testfalls <<CFM_System_aufrufen>> sind genaugenommen zwei Testschritte vonnöten: Softwaresystem „auswählen“ (aktivieren) und mit dem Auswählen wird auch der Applikationsserver zum Softwaresystem „gestartet“.

In der Abbildung illustriert Zeile 8 den Ablauf dieser beiden Testschritte. Die Funktion „systemUtil.Run“ öffnet einen Internet-Browser, gleichzeitig wird die Variable „psUrl“ mit übergeben, welche dann automatisch mit QTP ausgelesen wird. Die Variable „psUrl“ beinhaltet Informationen und die Adresse des Applikationsservers, wobei die Methode „Run“ die gleichzeitige analoge Ausführung dieser Funktion signalisiert. Zeile 14 überprüft, ob die in Zeile 8 initiierte Ausführung der Methode „Run“ erfolgreich war und ob dadurch das CFM-Softwaresystem gestartet werden konnte. Zeile 15 ruft implizit den nächsten Testschritt auf, unter der Voraussetzung, dass der Testschritt in Zeile 14 erfolgreich ist. Im gezeigten Testmodell sind diese Bedingungen durch die Kennzeichnung in den Bausteinen 2 und 3 dargestellt. Das Programmstück zwischen Zeile 8 und Zeile 17 in Abbildung 4.21 ist Bestandteil der Funktionsbibliothek „cfm_lib.vbs“ und wird im Funktionsbaustein „CFM_System_aufrufen“ hinterlegt. Zu Demonstrationszwecken werden an dieser Stelle die Funktionsbibliothek und gleichzeitig die Funktionsbausteine dargestellt.

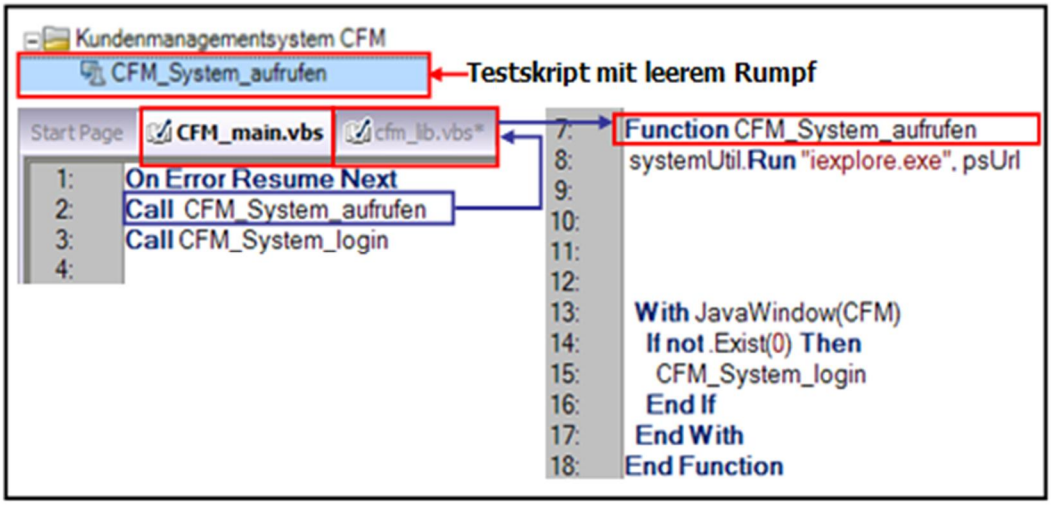


Abbildung 45: Vom Testskriptrumpf zum fertig ausführbaren Testskript

Durch den angefertigten Mechanismus („Adaptierung“) ist die Verbindung zwischen dem vom Testfallgenerator generierten Testskriptrumpf und den in den Funktionsbibliotheken hinterlegten, korrespondierenden, implementierten Testschritten vorhanden. Beim Aufruf des Befehls „Call CFM_System_aufrufen“ aus dem Hauptprogramm (CFM_main.vbs) wird dann die in der Funktionsbibliothek hinterlegte Testablauflogik (bspw. Zeile 8–17) ausgeführt. Zu jedem Softwaresystem wird analog ein Hauptprogramm angelegt, woraus die Funktionen später wie nach einem Baukastenprinzip aufgerufen werden können [77].

Im Rahmen dieser Dissertation kann die Reihenfolge eines Testablaufs entweder in einem Testskript definiert (Anordnung der Funktionen) oder durch einen Testscheduler im HPQC [131], [161] festgelegt werden. Beide Alternativen sind Bestandteil dieser Dissertation, wobei der Testscheduler zusätzlich den Vorteil der Zeitscheibe bietet. So werden Testfälle nach zeitlichen Vorgaben angeordnet und nach einer Zeitscheibe ausgeführt. Auch ist

der Testscheduler für die spätere Bündelung der Testskripte zu einer lauffähigen Testsuite verantwortlich [139]. Eine Testsuite besteht aus mehreren Testsets.

Nachdem die Abarbeitung der Funktion „CFM_System_aufrufen“ abgeschlossen ist, soll die „CFM-Anmelderoutine“ (Baustein 3) erläutert werden. Zeile 3 des Hauptprogramms ruft die Funktion für die Anmelderoutine auf, anschließend werden die dort implementierten Testschritte vom Testautomaten interpretiert und ausgeführt.

Der Unterschied zwischen <<CFM_System_aufrufen>> und <<CFM_System_login>> liegt darin, dass die Anmelderoutine zum Ausführen der Testschritte Testdaten benötigt. Testdaten können auf zwei Wegen erfasst werden, zum einen in UML-Klassendiagrammen und zum anderen in Exceltabellen. Der Autor dieser Dissertation kombiniert beide Wege, so werden die Testdaten in UML-Klassendiagrammen zunächst mit Hilfe von Fachbereichen spezifiziert. Mit UML-Klassendiagrammen bietet es sich an, Testdatentypen mittels Äquivalenzklassen zu strukturieren. In Klassendiagrammen können Datentypen bis zu finalen Äquivalenzklassen verfeinert werden. Auch werden die einzelnen Operationen in den Klassendiagrammen spezifiziert. Diese Klassendiagramme können, wenn benötigt, zu jedem Teststep im Testmodell angehangen werden. Analog dazu können auf Basis von Klassendiagrammen Testdaten direkt im Excelformat spezifiziert werden.

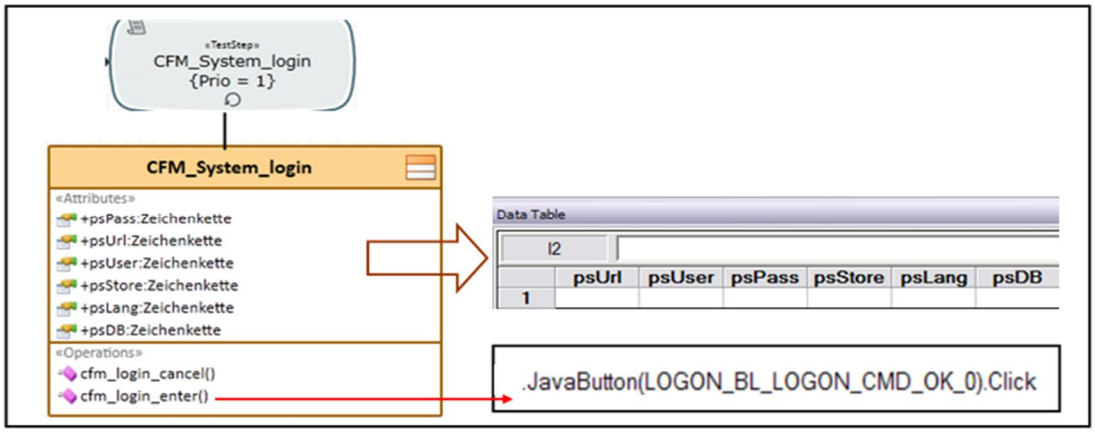


Abbildung 46: BPMN-UML-Klassendiagramm für Testdaten

Diese spezifizierten Dateien werden dann bei der späteren Testausführung automatisiert vom Testautomatisierungswerkzeug eingelesen und die darin enthaltenen Testdaten werden im Testlauf verarbeitet. Vor der Testdurchführung müssen noch fehlende Datenwerte ergänzt werden. Dazu gehören typischerweise die Sollwerte, da es im Modell keine Möglichkeit gibt, diese abhängig von den Eingabewerten berechnen zu lassen. Eine erprobte Variante ist es, die Sollwerte als erwarteten Wert direkt in die Testdaten zu integrieren [77]. Ein während der Laufzeit ermittelter Wert kann somit direkt mit dem in den Testdaten hinterlegten Sollwert verglichen und ausgewertet [8] werden. Der Vorteil dieser Vorgehensweise ist, dass Werte schon während der Laufzeit mit Vergleichsfunktionen aus den Funktionsbibliotheken kombiniert werden. Generell bietet die Erfassung von Testdaten, bspw. im Excelformat (als Tabellen), die Möglichkeit, einfach Ergänzungen an Testdaten durchzuführen oder eine zusätzliche Testvariante hinzuzufügen. So kann der Testfall <<CFM_System_login>> mit verschiedenen Marktnummern und Datenbankschemen durchgeführt werden. Änderungen aufgrund von Erweiterungen von Märkten oder auch Benutzern können leicht aufgenommen werden.

Mit dem Testschritt <<CFM_System_login>> werden sowohl die Datei „Testdaten_CFM_login“ als auch das dazu passende UML-Klassendiagramm mit den Spezifizierungen der Testdaten referenziert.

```
20  Function CFM_System_login(psUrl, psUser, psPass, psStore, psLang, psDB)
21   CFM_App_close
22     cfm_app_login = false
23     systemUtil.Run "iexplore.exe", psUrl
24
25  If Not JavaWindow(CFM).JavaInternalFrame(LOGON).Exist(60) Then
26        reporter.ReportEvent micFail, "Start", "No Login screen"
27        ExitRun(0)
28     End If
29     reporter.ReportEvent micPass, "Start", "Login screen available"
```

Der große Vorteil ist, dass die für die Tests benötigten Testdaten direkt von QTP eingelesen werden können. Dazu werden folgende Funktionen zusätzlich angefertigt:

Public Function qc_ImportSheet (psFolder, psAttName, psScrSheet, psDesSheet)

Diese Funktion ist Bestandteil der Funktionsbibliothek und übernimmt den Import aller externen Dateien, welche Testdaten für den spezifizierten Testfall beinhalten. Die Daten werden immer während der Testlaufzeit zum passenden Testskript geladen. Auch können ermittelte Testergebnisse wiederum direkt in die Testdaten integriert werden.
Da diese Funktionen einen generischen Charakter aufweisen, können sie von beliebigen zu

```
32   DataTable.AddSheet "CFM_System_login"
33      qc_ImportSheet "", "CFM_Testuser.xls", "CFM_System_login"
34              RowCount =DataTable.GetSheet("CFM_System_login").GetRowCount
```

testenden Softwaresystemen genutzt werden. So werden damit sowohl Testdaten von einem CFM-System in QTP geladen als auch Testdaten zum SAP-System. Analog dazu können mit der Funktion:

Public Function qc_att_tstest (psAttachName, psAction)

Testdaten und Testergebnisse aus einem Testlauf ausgelesen und wiederum in Exceltabellen abgelegt werden. Es kann auch direkt in die eben geöffnete und vom Testautomaten eingelesene Datei „hineingeschrieben“ werden. Diese beiden Funktionen sind für das hier zu betrachtende Fallbeispiel zwar nicht von entscheidender Bedeutung, doch bilden sie die Brücke zwischen dem Beschaffen von Testdaten und umgekehrt. Daher bilden beide Funktionen zwei wichtige Aktivitäten in einem Test ab, nämlich das Laden von zuvor erfassten Testdaten aus externen Dateien und das Sichern von zur Laufzeit generierten Testdaten wiederum in externe Dateien. Diese beiden Funktionen sind auch Bestandteil der Funktionsbibliotheken und haben globalen Charakter, da sie von allen beteiligten Testskripten genutzt werden können. Analog dazu können XML-Dateien Testdaten bereitstellen. Auch eine solche Funktion ist Bestandteil der Funktionsbibliothek, damit können Testdaten, wenn gewollt, ebenfalls mit Hilfe von XML strukturiert werden.

```
36      With JavaWindow(CFM).JavaInternalFrame(LOGON)
37
38       psUser = DataTable("psUser","CFM_System_login")
39       psPass = DataTable("psPass","CFM_System_login")
40       psStore = DataTable("psStore","CFM_System_login")
41       psDB = DataTable("psDB","CFM_System_login")
42       psLang = DataTable("psLang","CFM_System_login")
43
44        .JavaList(LOGON BL LOGON LIST LANG 0).Select psLang
```

Zeile 20 in der Funktionsbibliothek „cfm_lib.vbs“ führt die Funktion aus, welche aus dem Hauptprogramm CFM_main.vbs aufgerufen wird. In dieser Funktion sind sämtliche Schritte programmiert, welche eine Anmelderoutine erfordern. In den Variablen (psUrl, psUser, psPass, psStore, psLang, psDB) werden die für den Test erforderlichen Testdaten abgelegt. In der Variablen „psUrl“ wird die Adresse des Applikationsservers hinterlegt. Da Kundenmanagementsysteme über Clients bedient werden, ist die Adresse des Applikationsservers notwendig. Der Benutzername und das jeweilige Passwort sind in den Variablen „psUser“ und „psPass“ abgelegt. Die jeweilige Marktnummer, die Sprache des Softwaresystems, kann über die Variablen „psStore“ und „psLang“ spezifiziert werden. Bei Kundenmanagementsystemen werden die verschiedenen Daten in Datenbankschemas gespeichert. Mit der Variablen „psDB“ kann sowohl der Servername als auch das Datenbankschema mit angegeben werden.

Data Table

	psUrl	psUser	psPass	psStore	psLang	psDB
1						

Abbildung 47: Testdaten Excelformat

Die Auslagerung der Testdaten im Excelformat ermöglicht es, Varianten anzulegen. Einmal festgelegte Erweiterungen und Ergänzungen in Testdaten, bspw. durch Fachbereiche, können immer wieder verwendet werden. Diese sind für den Regressionstest essentiell [1]. Das untere Programmstück 47–54 übernimmt die Zusicherung für eine Java Session, das CFM basiert auf Java. Solche Zusicherungen werden wiederum zu einer „Settingbibliothek“ zusammengefasst, diese beinhaltet administrative Skripte, welche die Ausführung der Tests erst ermöglichen. Dazu gehören Initialisierungen von Testumgebungen (Testen von Verfügbarkeiten von Applikationsservern) oder das Bereinigen von „Cookies“ oder „Browsersitzungen“, damit ein neuer Test gestartet werden kann.

```
46       'Grant Java for session
47       With JavaDialog(JAVA_SECURITY_JD)
48          If .Exist Then
49             .JavaButton(JAVA_SECURITY_JB_Grant_Session).Click
50          End If
51       End With
52            .JavaButton(LOGON_BL_LOGON_CMD_OK_0).Click
53    End With
54 End Function
```

Sowohl bei einer erfolgreichen als auch bei einer nicht erfolgreichen Anmeldung muss eine Auswertung der Testschritte erfolgen, welche zentral im HPQC dokumentiert und verwaltet werden. Auswertungen von Testschritten und Testergebnissen werden zum späteren Nachweis für den Kunden benötigt. Auch können solche Auswertungen gesetzliche Hintergründe haben [142]. Im Testmodell „CFM_Anmelderoutine“ zeigt Baustein 4 analog die korrespondierenden Verifikationspunkte.

Zeile 57 prüft den im Testmodell definierten Verifikationspunkt <<CFM_Login_anzeige>>. Solange die „CFM-Anmeldemaske“ erscheint, werden die Funktionen in den Zeilen 63, 65, 66 und 67 <<CFM_System_login>> ausgeführt. In beiden Fällen, also sowohl im „IF“- als auch im „ELSE“-Zweig, werden die jeweils benötigten Auswertungsmeldungen generiert.

Eine Eigenschaft von CFM-Systemen ist, dass solche Systeme auf Datenbankschemas basieren. Analog dazu arbeiten SAP-Systeme mit verschiedenen Mandanten. Damit zum

```
56  'Report if Login successfull
57     If JavaWindow(CFM).JavaInternalFrame(LOGON).Exist(5) Then
58
59        cfm_report_event "loginfail", ""
60        'ExitRun(0)
61     Else
62        'reporter.ReportEvent micPass, "Login OK", "Main Form available"
63        cfm_report_event "loginpass", ""
64
65        cfm_proceedErrorDlg false
66          cfm_proceedErrorMsg false
67        CFM_System_login = true
68     End If
```

Testzeitpunkt mit dem „korrekten“ Datenbankschema getestet werden kann, muss in der Variablen „psDB“ das für den Test vorgesehene Datenbankschema angegeben sein.

Dieser Abschnitt demonstrierte, wie aus einem Testmodell ein generierter Testfall und ein fertig ausführbares Testskript entstehen können. Wobei nicht nur die Adaptierung, sondern auch die notwendigen Namenskonventionen der Testfälle und Funktionen jeweils im Vordergrund stehen. Das Zusammenspiel der jeweiligen Funktionen und der korrespondierenden Funktionsbibliotheken wurde am Beispiel der Testfälle <<CFM_System_aufrufen>> und <<CFM_System_login>> dargestellt. Zusätzlich muss bei der Spezifikation von Testschritten beachtet werden, dass die Testschritte und die dazugehörigen Testdaten unmittelbar Einfluss auf das Testergebnis haben. So sind immer Äquivalenzklassen zu bilden, um auch die Randfälle zu testen. Aber auch Negativtestfälle müssen modelliert werden, um sowohl das Verhalten des zu testenden Systems als auch gleichzeitig die Qualität und die Güte der Testfälle zu testen.

3.7.1.3 Funktionsbibliothek Kunden anlegen – Testdatenspezifizierung

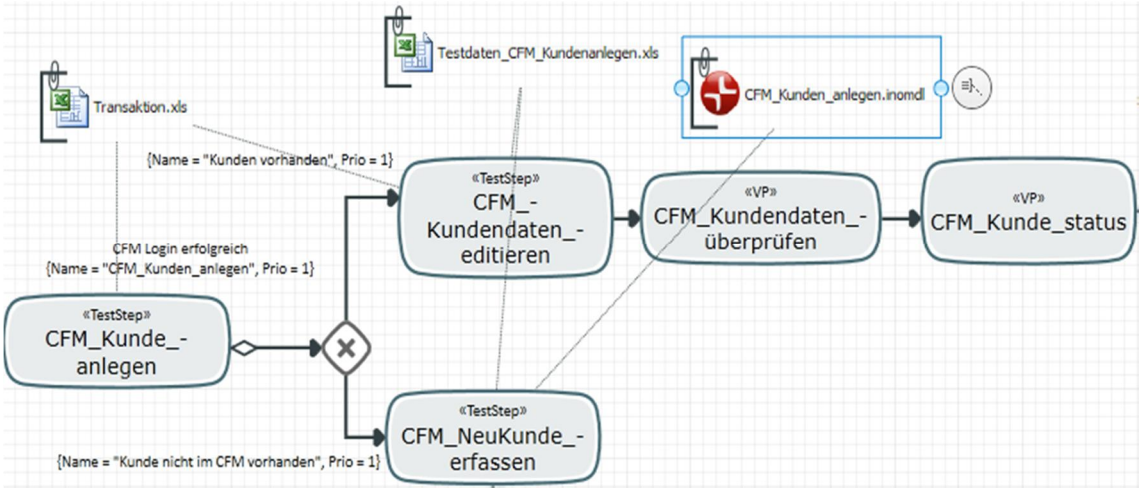


Abbildung 48: Ausschnitt Testmodell

Sobald über die Anmelderoutine das Anmelden am CFM-System erfolgt, rücken die Aktivitäten in den Vordergrund, welche für die Kundenanlage relevant sind. Mit Hilfe der zuvor konfigurierten und erweiterten Stereotypen in BPMN können jetzt die Aktivitäten als Testschritte im Testmodell gekennzeichnet und zusätzlich mit einem Testfallnamen und einer Priorisierung versehen werden. Aus der Abbildung können insgesamt 3 Testfälle: <<CFM_Kunde_anlegen>>, <<CFM_Kundendaten_editieren>> und <<CFM_Neukunde_erfassen>> sowie zwei Verifikationspunkte: <<*CFM_Kundendaten_überprüfen*>> und <<*CFM_Kunde_status*>> in Form von „Checkpoints“ abgeleitet werden. Sowohl das Editieren der Kundendaten als auch die Erfassung von Kundenstammsätzen erfordern die zuvor festgelegten Testdaten. Analog zum Vorkapitel erfolgt die Testdatenmodellierung in UML-Klassendiagrammen und anschließend werden diese Daten im Excelformat weiter spezifiziert und um Varianten auf der Basis von Äquivalenzklassen erweitert.

Jeder dieser einzelnen Testschritte kann aus einem oder mehreren Testschritten bestehen. Damit diese Schritte nicht manuell ausgeführt werden müssen, werden diese direkt in Funktionen implementiert und in vorhandene Funktionsbibliotheken integriert. Mit dem Testfall <<CFM_Kunde_anlegen>> kann mit der Implementierung dieser Testschritte begonnen werden. Anschließend sorgt die zuvor angefertigte Adaptierung mit dem Befehl „Call“ dafür, dass dieser Test nach der Generierung aus dem Testmodell auf ein korrespondierendes Testskript in der Funktionsbibliothek referenziert. Ist bis dato keine Funktion in der Bibliothek vorhanden, so fordert das HPQC mit einer Meldung dazu auf, diese Funktion anzulegen und zu implementieren.

Das CFM-System wird, genauso wie das SAP-System, durch das Aufrufen von verschiedenen Transaktionen, welche wiederum aus Programmen, Reports und/oder Formularen bestehen, ausgeführt. Im Testmodell sind im Testschritt <<CFM_Kunde_anlegen>> die Dateien sowohl für die Testdaten im Excelformat als auch die Spezifizierungen in UML-Klassendiagrammen hinterlegt. Die Programme, Formulare oder Reports, welche über Transaktionen aufzurufen sind, werden auch als Bestandteil der Testdaten betrachtet. So können diese über die jeweiligen Namen (der Programme, Transaktionen oder Reports) aufgerufen werden. Mit dem Aufruf der Transaktion zur Neukundenanlage fragt das Sys-

tem explizit, ob es sich hierbei um eine Neuanlage oder um die Bearbeitung eines Bestandskunden handelt. Bei der Testfallmodellierung können noch weitere Varianten spezifiziert werden, bei der Generierung der Testfälle können sowohl für die Neuanlage als auch bei der Bearbeitung eines Bestandskunden jeweils die Testfälle mit mehreren Varianten ausgeführt werden. Eine mögliche Variante ist, dass ein Bestandskunde sein Kundenkonto, welches durch einen Zahlungsausfall gesperrt ist, wieder entsperren möchte. Dazu werden Daten aus dem SAP-System erwartet, welche die Zusatzinformation (Debitoren entsperren) enthalten.

Im Fallbeispiel dieser Dissertation soll ein neuer Bestandskunde erfasst werden. Dazu muss die CFM-Transaktion „K00F09“ im Eingabefenster des Einstiegsbildschirms vom CFM-

```
65  Function data_get_from_sheet(psFolder, psAttName, psScrSheet, psParameter)
66          Dim  TCFound
67          TCFound = False
68          DataTable.AddSheet psScrSheet
69          qc_ImportSheet psFolder, psAttName, psScrSheet, psScrSheet
70              RowCount =DataTable.GetSheet(psScrSheet).GetRowCount
71  End Function
72
73
74  Function cfm_get_tab(sFormID, sID)
75    Select Case sFormID
76    Case "K00F01"
77      Select Case sID
78
79       Case "A"
80       cfm_get_tab = cfm_get_tab_prompt(sFormID, "BL_CTRL", "TAB_IDENTIFICATION") '
81          cfm_get_tab_alt = "Identification" '3
82        Case "B"
83          cfm_get_tab = cfm_get_tab_prompt(sFormID, "BL_CTRL", "TAB_ADDRESS") '
84          cfm_get_tab_alt = "Address" '4
85
86          ....
87
88           Case Else
89        cfm_report_exception "cfm_menu: Transaktion nicht verfügbar: " & currentMenu
90        cfm_form_ctrl = -1
91        Exit function
92    end select
93
94  Tab_Overview_Kunden_Name= DataTable("Vorname", dtGlobalSheet)
95  Tab_Overview_Kunden_Nachname= DataTable("Nachname", dtGlobalSheet)
96  .JavaButton(SAVE BL SAVE CMD OK 0).Click
```

System eingegeben werden.

Die Transaktionsnamen stehen immer für einen im Hintergrund zu initiierenden Geschäftsprozess. So kann eine Transaktion mehrere Unterprozesse initiieren, diese wiederum besitzen für jeden Testschritt eine eigene Testdatendatei. Genau diese Komplexität macht es schwierig, diese Prozesse End-to-End zu testen.

Das obige Programmstück zeigt, welche Programmstücke hier miteinander agieren, Zeile 69 ruft die globale Funktion für das Einlesen der für den Testschritt <<CFM_Kunde_anlegen>> hinterlegten Testdaten auf. Zeile 74 ist die generische Funktion, welche hier nur in einem kleinen Ausschnitt dargestellt wird. Je nachdem, welche CFM-Transaktion hier als Variable übergeben ist, kann mit Hilfe der „Case“-Anweisung

genau für die übergebene Transaktion der korrespondierende Funktionsblock aufgerufen werden, welcher die programmierten Testlogiken enthält. So ist es im Grunde egal, welchen Testpfad der Testfallgenerator aus dem Testmodell generiert, es steht immer die passende, fertig implementierte Testablauflogik zu dieser Transaktion bereit. Sollte eine Funktion fehlen, so weist eine Meldung darauf hin, dass eine passende Funktion zu diesem Testskript implementiert werden muss.

Transaktion „K00F01“ ist für den Aufruf des Kundenformulars zuständig. Sobald nach dem Anmelden der Einstiegsbildschirm erscheint, kann mit der Pflege der Kundendaten begonnen werden. Die beiden Zeilen 94 und 95 im obigen Programmstück zeigen einen kleinen Ausschnitt, wie Testdaten in das CFM-System mit Hilfe des angefertigten Testskriptes gelangen. Dazu werden zur Laufzeit die hinterlegten Testdaten ausgelesen und eingepflegt. Zeile 96 illustriert nur ein kleines Beispiel, wie nach erfolgreichem Pflegen der Testdaten der „OK“-Button in der Anwendung „gedrückt“ wird. Mit Hilfe dieser Funktion können jetzt beliebige Buttons aus dem CFM-System hinzugefügt werden, wie bspw. „Abbrechen“, „Speichern“ usw.

Prüfpunkte sind hier beispielsweise in Zeile 99 hinzugefügt, wo eine Exception ausgelöst werden soll, wenn eine eingegebene Transaktion nicht verfügbar ist. Auch können an beliebigen Stellen im Programm Verifikationspunkte in Form von Prüfroutinen hinzugefügt werden.

```
99  Public function cfm_form_exist(psFormName)
100   cfm_form_focus = cfm_form_ctrl (psFormName,2,True)
101 End Function
```

Mit der Funktion in Zeile 99 kann überprüft werden, ob eine eingegebene Transaktion tatsächlich existiert. Sollte es hier während des Testlaufs zu einem versehentlichen Tippfehler gekommen sein, so muss das System mit einer „Exception“ reagieren. Auch diese Exceptions sind ein Teil der Teststrategie und werden zu Testzwecken während eines Testlaufs bewusst provoziert, um zu überprüfen, ob das System darauf reagieren kann. Warnungen und Fehlermeldungen werden mit den beiden globalen Funktionen:

Public Function cfm_proceedErrorMsg (pbResult)

Public Function cfm_proceedErrorDlg (pbResult)

abgefangen, gleichzeitig übernehmen die Funktionen das Generieren eines Fehlerreports im HPQC. Der Vorteil dieser beiden Funktionen ist, dass sie global und dadurch auch durch alle anderen zu testenden Softwaresysteme genutzt werden können. Fehlerreports sind nach jedem Testlauf durch einen Tester oder Fachbereich zu betrachten, um den Testverlauf nachzuvollziehen. Eine zusätzliche Funktion:

Public Function cfm_capture_main

sorgt für ein besseres Verständnis sowohl beim Fachbereich als auch bei den Testern. Mit dieser Funktion kann zu jedem Testschritt ein Bildausschnitt angefertigt werden, welcher dann zu dem Testauswertungsreport am Ende eines Testlaufs hinzugefügt wird. So kann nach jedem Testlauf der Testauswertungsreport betrachtet werden, um bspw. zu prüfen, an welchen Stellen im Prozess das System Warnungen oder Fehlermeldungen signalisierte. Im Fehlerfall muss nach eingehender Analyse durch einen Fachbereich oder einen Entwickler ein Fehler auch im Testmanagementtool erfasst werden. So bleibt die Rückverfolgung von

Testanforderung, Testmodell und generiertem Testskript bis zum ausgeführten Test und bis zu den resultierenden Testergebnissen erhalten. Zum Vergleichen von Testergebnissen können folgende Funktionen genutzt werden:

```
Public Function cfm_status_check (psText, psBool)

Public Function cfm_main_result_check
```

Mit diesen beiden generischen Funktionen lassen sich beliebige Testergebnisse auslesen und vergleichen. Damit können die im Testmodell hinterlegten Verifikationspunkte direkt durch Funktionen abgebildet werden. Ein Verifikationspunkt, der einen Prüfpunkt als Test generiert, kann somit direkt auf einen Funktionsbaustein referenzieren. Auch bietet dies die Möglichkeit, Testergebnisse für einen weiteren Test zur Verfügung zu stellen. So bietet die Funktion „cfm_main_result_check“ Mechanismen, die es erlauben, Testergebnisse im Excelformat zu erfassen und somit Testdaten unmittelbar nach der Generierung direkt an einen weiteren Testfall, welcher von den Testergebnissen des vorangegangenen Testfalls abhängig ist, zu übergeben. Eine generierte Kundennummer kann auf diesem Weg direkt in die Testdatendatei des Kreditmanagementsystems, des Kassensystems und der Buchhaltungssoftware SAP-FI geschrieben werden. Nachdem gezeigt wurde, wie in einem CFM-System ein Kunde generiert wird und welche Aktionen sich hinter einem konkreten Testschritt befinden, werden jetzt die angebundenen Systeme näher beleuchtet.

3.7.2 Testfallmodellierung Kreditsystem

Ein Kundenmanagementsystem sendet Daten zu angebundenen Schnittstellensystemen, wie bspw. einem Kreditsystem. Das Kreditsystem ist ein komplexes Gebilde, welches auf mathematischen und statistischen Verfahren basiert. Allein diese Applikation in „Gänze“ zu testen, erfordert einen sehr hohen Ressourcen- und Zeitaufwand [142]. Zumal der Einsatz von Äquivalenzklassen und Grenzwertanalysen an dieser Stelle zwingend notwendig ist, um die in Kreditsystemen implementierten Regelwerke mit allen möglichen Kombinationen zu testen. Auch arbeiten Kreditsysteme nach gleichen Mustern [142]. So sind, je nach konfigurierter Unternehmensrichtlinie, bis zu 1200 Testfälle im Standardpaket solcher Softwaresysteme zu testen [159], [154]. Für den Autor dieser Dissertation dient das Kreditsystem „nur“ als Blackbox. D. h., es ist ein Teilsystem in einer Kette von Softwaresystemen, welches dazu beiträgt, einen Geschäftsprozess auszuführen.

Das Kreditsystem empfängt Stammdaten vom CFM, zusätzlich muss das Kreditsystem die erhaltenen Daten auf Vollständigkeit prüfen. Die Daten, die vom CFM zum Kreditsystem übertragen werden, sind folgende: (Vorname, Nachname, Geburtsdatum und Geburtsort, Anschrift und die vom CFM-System zur Laufzeit generierte Kundennummer). Im UML-Klassenmodell kann jetzt überprüft werden, ob die tatsächlich erforderlichen Daten auch das korrekte Format besitzen. Im UML-Klassendiagramm sind die Testdatenspezifizierungen hinterlegt. An dieser Stelle können bspw. Vorbedingungen im Modell dafür sorgen, dass Testdaten, bevor sie in ein Testsystem geladen werden, auf das korrekte Format geprüft werden. Da die Daten, die von einem Softwaresystem ins andere transferiert werden, in sogenannten SDIs auf Fileservern dieser Softwaresysteme abgelegt werden, kann jetzt ein Testskript, welches aus der Vorbedingungsaktivität entsteht, sicherstellen, dass die Da-

ten vollständig sind und das gewünschte Format besitzen. Erst dann werden diese in die Testsysteme geladen.

3.7.2.1 Schnittstellendaten

Schnittstellendaten können neben dem Excelformat auch in XML strukturiert und auf den Fileservern zur weiteren Verarbeitung abgelegt werden. Aber auch Schnittstellendaten in Form von „Textdateien“ sind in der Praxiswelt üblich, daher muss zumindest für diese beiden Dateiarten auch eine adäquate Funktion existieren, welche diese Daten auf das korrekte Format und den korrekten Inhalt prüft. Die Funktion:

Function Public download_testdata_XML

sorgt dafür, dass sowohl Testdaten vom Fileserver als auch vom Testsystem in einer XML-Struktur gespeichert werden. Diese Testdaten können mit einem speziellen „Mappingskript“, wenn erforderlich, in das Excelformat konvertiert werden. Anschließend erfolgt eine zentrale Speicherung im HPQC. Analog dazu gibt es die Funktion, welche dann den automatisierten Download der Testdaten zur Laufzeit übernimmt:

Function Public download_file_from_qc

Ein Testskript kann diese Funktionen nutzen, um die im Testmodell hinterlegten Vorbedingungen, wie bspw. die Testdatenvollständigkeit, auf Korrektheit zu prüfen.

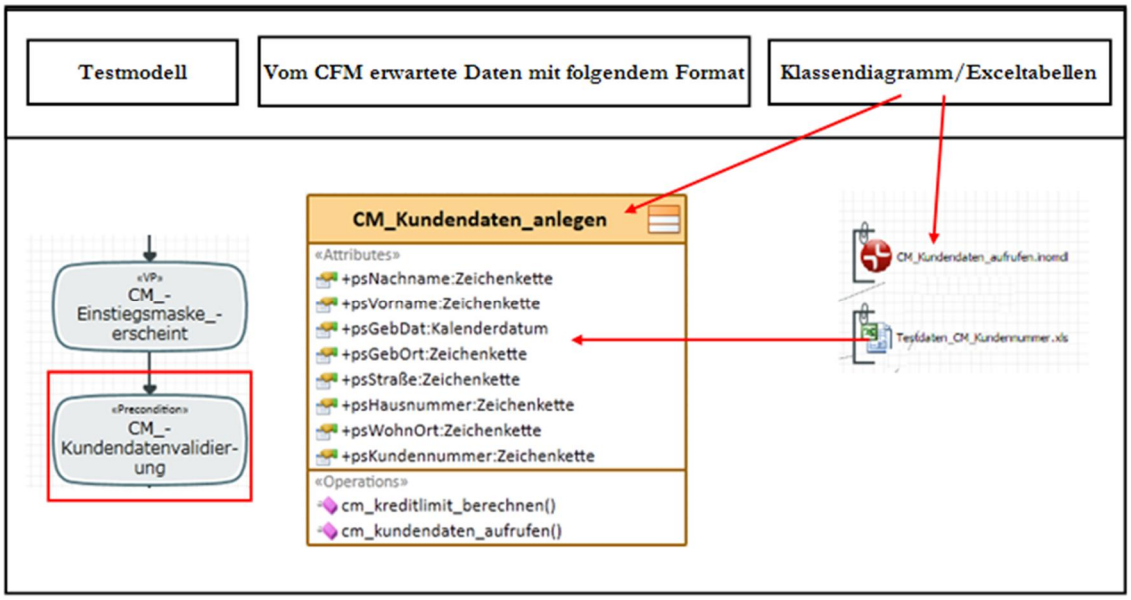


Abbildung 49: Verifikation der Testdaten im Testmodell

Dabei ist die Vorgehensweise immer gleich, zunächst „Abgreifen der Testdatei“, den „Inhalt auslesen“, die dort hinterlegen Parameter mit den spezifizierten Parametern im UML-Klassendiagramm und in den Exceltabellen vergleichen.

Ein Beispiel, bezogen auf die Schnittstelle zwischen CFM und Kreditsystem, zeigt, welche Daten eine über diese Schnittstellen übertragene Datei enthalten darf und welches Format diese Daten besitzen müssen [157]. Kommen Parameter hinzu, so müssen diese explizit sowohl im Klassendiagramm als auch in der Prüfroutine erweitert werden. Ist das nicht der Fall, so wird der Testlauf immer daran scheitern, dass die Testdaten nicht vollständig sind. Sprich, die Vorbedingung der Testdatenvollständigkeit ist nicht gegeben [149].

Die hier als Vorbedingung modellierte Aktivität <<CM_Kundendatenvalidierung>> übernimmt die oben beschriebenen Schritte. Zunächst werden die Testdaten vom Fileserver des jeweiligen Applikationsservers abgegriffen, dann auf Vollständigkeit überprüft. Als Beispiel erwartet die Schnittstelle zwischen Kreditsystem und CFM nur die in der Abbildung 4.26 aufgeführten Attribute und deren Parameter. Es darf weder ein Attribut noch eine Parametrisierung fehlen, umgekehrt dürfen keine Attribute zusätzlich „einfach“ hinzugefügt oder gelöscht werden. Genau diese Kriterien werden jetzt automatisiert durch das angefertigte Testskript überprüft.

Änderungen an den Schnittstellendaten erfordern eine Änderung in der Spezifikation dieser Softwaresysteme und daraus resultierend auch Änderungen in der Testspezifikation. An dieser Stelle lässt sich festhalten, dass sich mit Hilfe dieser Prüfungen frühzeitig Fehler in Testdaten identifizieren und beheben lassen [1]. Zusätzlich können Erweiterungen durch Änderungen an Spezifikationen leicht durchgeführt werden. Erweiterungen oder Änderungen an Testdaten können ohne großen Aufwand erfolgen. Es reicht der Zugriff auf die jeweilige Testdatei im Excelformat, um dann die notwendigen Anpassungen vorzunehmen.

3.7.2.2 Modellierung der Aktivität „Kreditsystem-CFM“

Der folgende kleine Ausschnitt aus dem Testmodell für das Kreditsystem weist eine Besonderheit auf. Er enthält eine Aktivität, welche auf das CFM-System verweist. Dies erleichtert die Testfallmodellierung, wenn Systeme miteinander verbunden sind, wie in diesem Beispiel. Durch die Einführung der Namenskonventionen ist es sowohl dem Testfallgenerator als auch dem Modellierungswerkzeug egal, welche Softwareeinheiten im Testmodell abgebildet werden. Bei der Generierung und der Adaptierung der Testskripte wird gleichzeitig die Reihenfolge der Testausführung anhand der Pfade vorgegeben. Sollte eine tatsächliche Anpassung der CFM-Daten vonnöten sein, so sorgt der Testfall <<CFM_Kundendaten_editieren>> dafür, dass die erfassten Stammdaten eines Kunden nachbearbeitet werden. Bei der Generierung von Testfällen über alle Pfade sollte dann aber auch sichergestellt werden, dass die Testdaten (denn diese müssen ja unvollständig sein), die dann zu einem Testpfad (CFM_Kundendaten_editieren) führen, auch tatsächlich für diesen Testfall unvollständig sind, damit dieser Pfad einmal als Testfall durchgespielt werden kann.

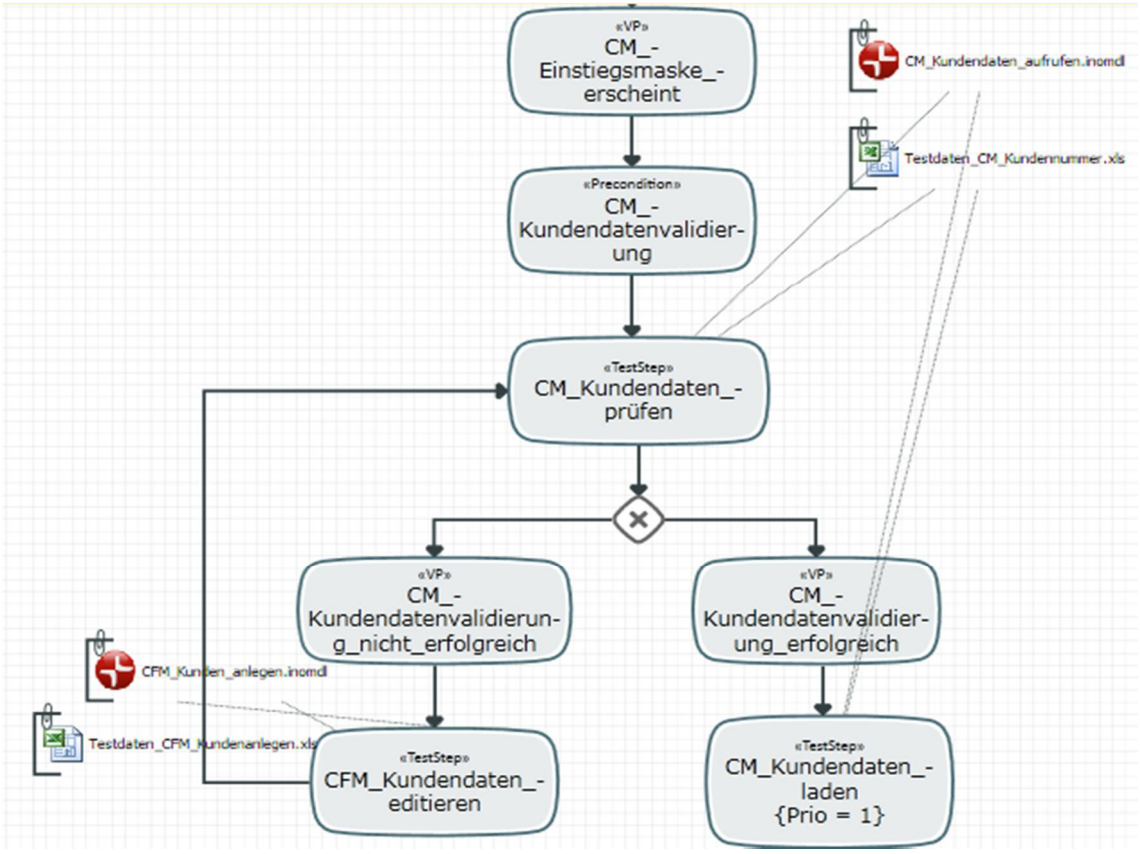


Abbildung 50: Testmodellausschnitt Kreditmanagementsysteme

Das Ergebnis des Kreditsystemprozesses ist am Ende die erfolgreiche Berechnung eines zu einem bestimmten Kunden angefragten Kreditlimits. Dieses Kreditlimit wird auf der Basis von historischen sowie soziologischen Daten ermittelt. Das Kreditsystem versorgt dann sowohl die Kassensysteme, das CFM als auch das SAP-FI-System mit tagesaktuellen Kreditinformationen zum jeweiligen Kunden. Aus Datenschutzgründen werden zu dem Kunden aus dem Kreditsystem nur die Kundennummer, das errechnete Kreditlimit sowie eventuelle Sperrvermerke übertragen.

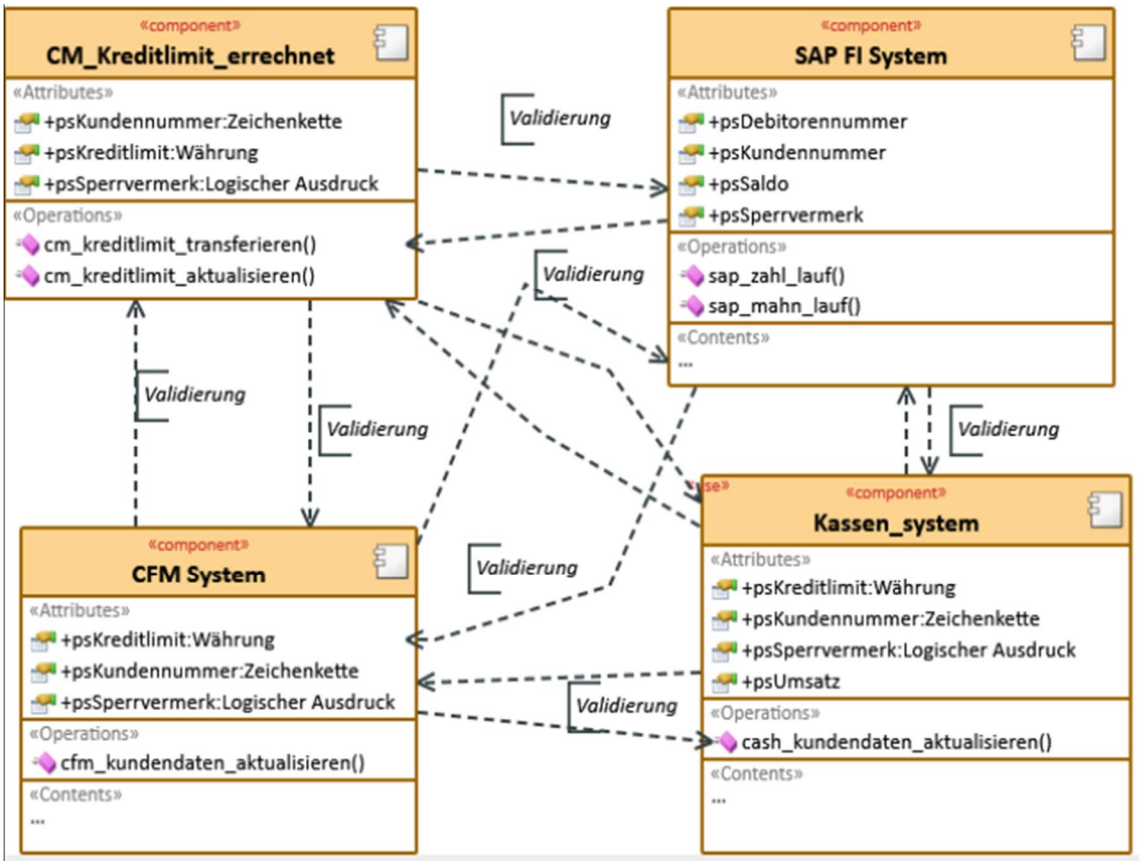


Abbildung 51: Schnittstellen und Testdatenvalidierung

Die Abbildung verdeutlicht, welche Daten zwischen dem Kreditsystem und dem angebundenen System übertragen werden, wobei der Datentransfer in beide Richtungen erfolgt. Die Validierung der Testdaten auf Korrektheit bzgl. Format und Vollständigkeit geschieht analog. Zum Einsatz kommen immer wieder die gleichen und schon einmal implementierten Funktionen aus den Funktionsbibliotheken.

Genauso prüfen die Testskripte die Verifikation (VP im Testmodell) der übergebenen Testdaten, sprich, wenn in der Schnittstellenspezifikation beschrieben ist, dass vom Kreditsystem zum SAP-System die Attribute (psKundennummer, psKreditlimit, psSperrvermerk) und umgekehrt vom SAP-System die Attribute (psKundennummer, psDebitorennummer, psSaldo, psSperrvermerk) ins Kreditsystem übertragen werden.

Die im Testmodell definierten Verifikationspunkte sorgen dafür, dass nur genau diese Attribute in den zu verarbeitenden Dateien enthalten sein dürfen. Die Funktion Function CM_Kundendatenprüfen übernimmt die Prüfung der Daten, welche vom CFM ins Kreditsystem übertragen werden, auf Format und Konsistenz. Die Zeilen 115–128 zeigen einen kleinen Ausschnitt aus der Prüfung.

```
114  ' Daten aus dem CFM
115  Function CM_Kundendatenprüfen(paTestdata)
116
117          Dim psKreditlimit,psKundenummer,psSperrvermerk
118
119          If  IsArray(paTestdata) Then
120                  psKreditlimit= paTestdata(0)
121                  psKundennummer= paTestdata(1)
122                  psSperrvermerk= paTestdata(3)
123
124          Else
125                  psKreditlimit=DataTable.Value("Kreditlimit","Initial")
126                  psKundennummer = DataTable.Value("Kundennummer","Initial")
127                  psSperrvermerk = DataTable.Value("Sperrvermerk","Initial")
128          End If
```

Daten aus dem CFM-System werden auch in die Buchhaltungssoftware SAP-FI übertragen, dort bekommen die erfassten Kunden jeweils eine mit der Kundennummer im CFM korrespondierende Debitorennummer, welche für die buchhalterischer Aktivitäten genutzt wird.

3.7.3 Testfallmodellierung Kassensystem

Kassensysteme empfangen von externen Systemen wie dem CFM Daten und senden bspw. Kassendaten an das SAP-FI-System. Das Kassensystem empfängt vom Kreditsystem errechnete Kreditlimits von Kunden und speichert diese in einem „speziellen“ Feld in der Kassensoftware. Der Grund hierfür ist, dass die Kreditverfügungen von Kunden so aktuell und schnell wie möglich verfügbar sein müssen. So kann bei der Eingabe der Kundennummer an der Kasse direkt durch den Kundenmanager eingesehen werden, wie hoch das jeweilige Kreditlimit zu einem bestimmten Kunden ist. Der Kassenprozess ist für den Gesamtprozess an dieser Stelle ein Datenlieferer, die Umsatzdaten eines Kunden sind ein wichtiger Bestandteil des Debitoren-Prozesses und bilden die Grundlage für den späteren Zahllaufprozess im SAP-FI.

Der große Vorteil ist, dass Kassensysteme, genauso wie alle anderen hier beteiligten Softwaresysteme, auf Applikationsservern laufen, so lassen sich jetzt die Daten über die vorhandenen Schnittstellen versenden oder empfangen. Schnittstellendaten können über Fileserver in die jeweiligen Softwaresysteme gelangen. Für den Test ist es notwendig, einen Zwischencontainer zu definieren, der Testdaten von einem Testfall zum anderen temporär speichert. Da sich für die Testdatenspezifizierungen Tabellen im Excelformat als bewährtes Mittel erwiesen haben, bietet es sich an, die temporär abzuspeichernden Daten auch in Exceltabellen anzulegen. Auch ist es wichtig, die Daten auf Konsistenz und Format zu prüfen.

3.7.3.1 Testfallmodellierung-Aktivität „Kassensystem-CFM“

In diesem Unterabschnitt soll gezeigt werden, wie generierte Kundennummern aus dem CFM-System für Testzwecke abgefangen und für den Testfall eines Kassensystems genutzt werden können. Abbildung 51 illustriert einen in BPMN-Testnotation modellierten Teilausschnitt eines Kassenprozesses.

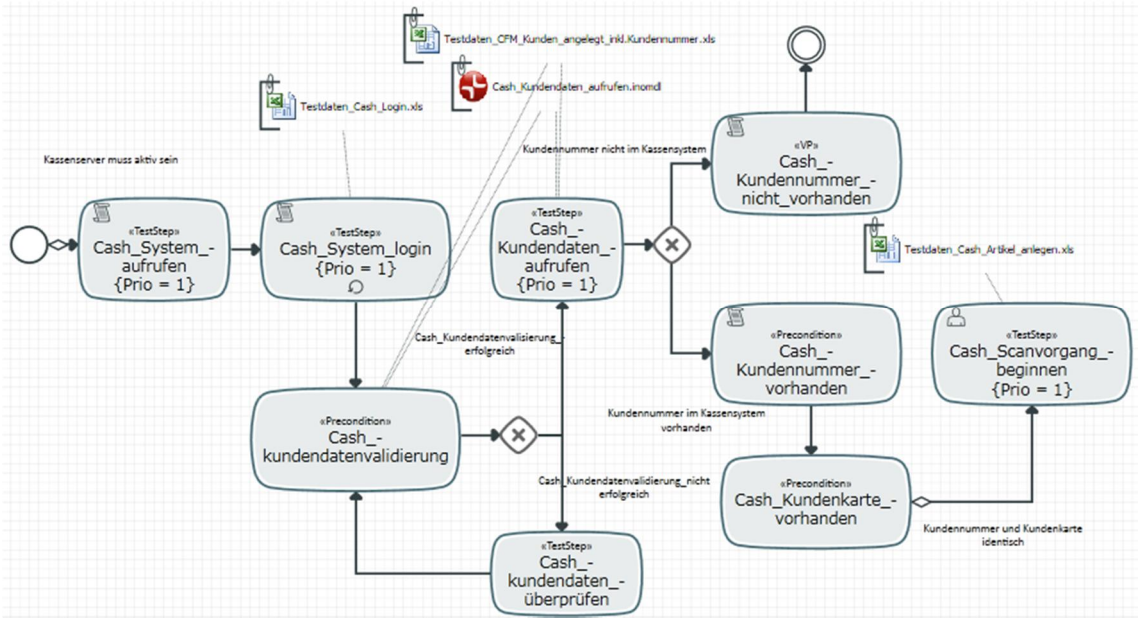


Abbildung 52: Teilausschnitt Testmodell Kassenprozess [Quelle Prozess: Wincor Nixdorf]

In der Praxis kann bspw. der Testschritt <<Cash_Scanvorgang_beginnen>> durch unterschiedliche Komponenten durchgeführt werden. In erster Linie wird der Scanprozess durch einen Kundenmanager durchgeführt, allerdings kann dieser für das Einscannen der Artikel und Produkte sowohl ein Lasergerät verwenden als auch die Daten per Artikelnummer direkt über eine Tastatur in das Kassensystem eingeben. Weitere Artikelerfassungsmöglichkeiten werden hier nicht betrachtet. Sowohl die korrekte Verarbeitung als auch das Anzeigen der jeweiligen Artikeldaten können direkt am Kassendisplay überprüft werden. Für die Testautomatisierung bietet es sich an, einen Scanvorgang durch die Eingabe der Artikeldaten über die Tastatur durchzuführen. So kann das Testen der Kassensysteme mit Artikeln simuliert und auch ohne manuellen Eingriff getestet werden.

Der Kassenprozess wird immer durch einen Kunden ausgelöst, der ausgewählte Artikel bezahlen möchte. Ein Kundenmanager muss sich dazu an einem Kassensystem (Client) anmelden. Dazu benötigt er die notwendigen Anmeldedaten (Benutzername und Passwort). Da sich die Anmelderoutinen bei den hier vorgestellten Softwaresystemen nur marginal unterscheiden, reicht es, wenn die zuvor implementierte Funktion zur Anmeldung am CFM-System herangezogen und minimal für die automatisierte Anmeldung am Kassensystem angepasst wird (bspw. Ersetzung der Softwarenamenskürzel usw.). Der Ausschnitt dieses Testmodells illustriert, welche Testschritte, Vorbedingungen und Verifikationspunkte vonnöten sind, um einen Kunden über das Kassensystem aufzurufen.

Die Anmeldedaten werden, wie schon beim CFM-System demonstriert, automatisiert mit der Funktion „Function qc_ImportSheet(psFolder, psAttName, psScrSheet, psDesSheet)“ ausgelesen. Die dort hinterlegten Testdaten werden direkt in das Kassensystem übernommen und eingegeben.

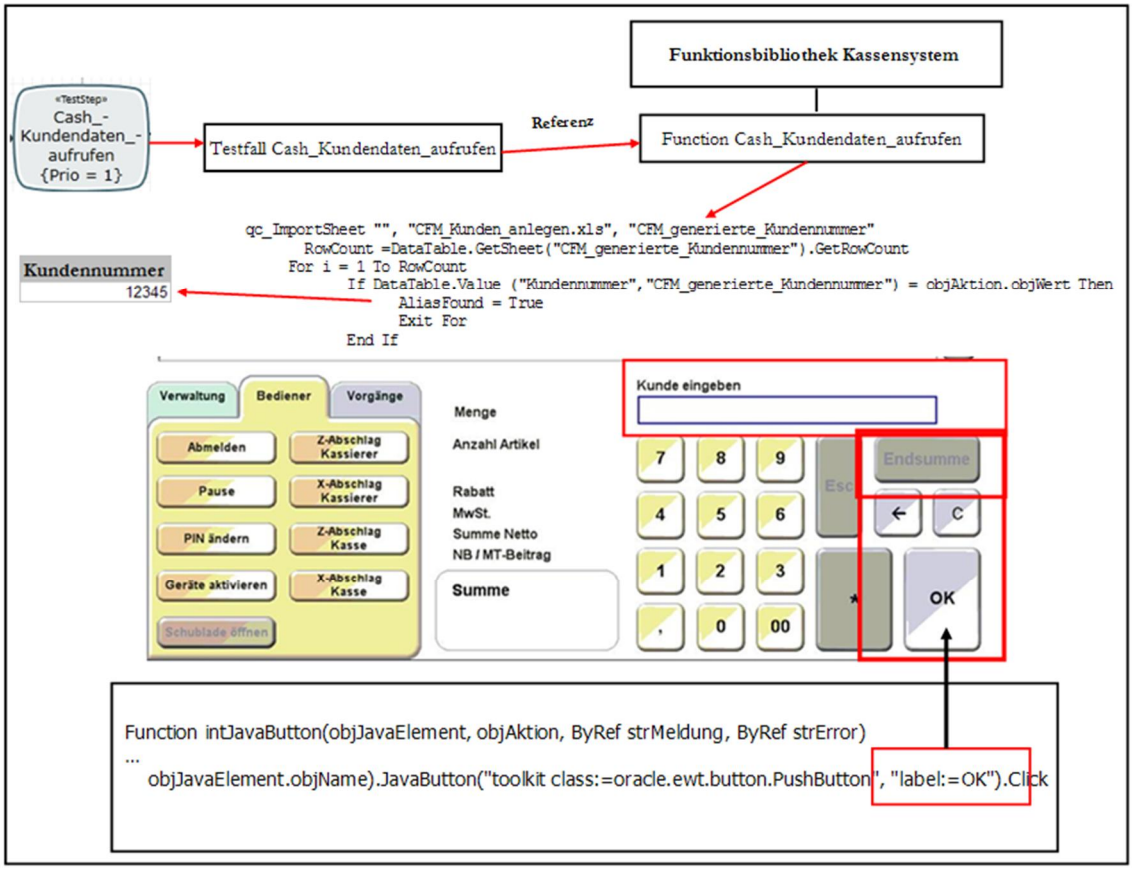


Abbildung 53: Kassenoberfläche Wincor Nixdorf

Als nächster Schritt ist das Aufrufen der Kundendaten des jeweiligen Kunden durchzuführen. Hier muss ein Zugriff auf die Testdaten, welche das CFM zuvor generiert hat, erfolgen. Das CFM hat zuvor bspw. einen Kunden mit der Kundennummer „12345“ angelegt und in den globalen Exceltabellen hinterlegt. Durch das HPQC und die dort festgelegten Ordnerstrukturen können jetzt beliebige Testskripte wieder auf diese Informationen und Daten zurückgreifen. Der Testschritt <<Cash_Kundendaten_aufrufen>> besteht, nachdem sich der Kundenmanager am Kassensystem angemeldet hat, aus zwei Schritten: die Identifizierung der Kundennummer (Eingabebox) und das Drücken des „OK“-Buttons am Kassendisplay. Die Abbildung illustriert den oben beschriebenen Testablauf im Testschritt <<Cash_Kundendaten_aufrufen>>.

Begonnen wird mit der Generierung des Testschritts, analog dazu erfolgt die Referenz auf eine konkret hinterlegte Funktion, welche genau diesen konkreten Testschritt ausführt. Eine weitere Funktion fasst alle Javaobjekte zusammen, damit bspw. Buttons jeder Art, jeder Funktion und mit jeder Beschriftung durch ein Testwerkzeug identifiziert und ausgeführt werden können. Ein Beispiel ist das Drücken des Buttons „OK“, nachdem eine Kundennummer im Feld „Kunde eingeben“ eingepflegt wird. Anschließend erscheint der ausgewählte Kunde auf dem Kassendisplay. Für den Fehlerfall müssen die zuvor schon für das CFM-System definierten Standardfunktionen die Fehlerbehandlung übernehmen. Sollte der soeben im CFM angelegte Kunde aufgerufen werden können, so ist damit auch der Schnittstellentest gelungen. Denn die Kundendaten werden über die CFM-Schnittstelle an die Kassensysteme übertragen. Sowohl auf den Scanvorgang als auch auf die Prüfung der Artikel im Kassendisplay kann an dieser Stelle verzichtet werden. Der nächste Testschritt,

welcher in Betracht gezogen werden soll, ist dafür verantwortlich, das Kassensaldo und somit den Kassenabschluss an die Buchhaltungs-SAP-FI zu transferieren.

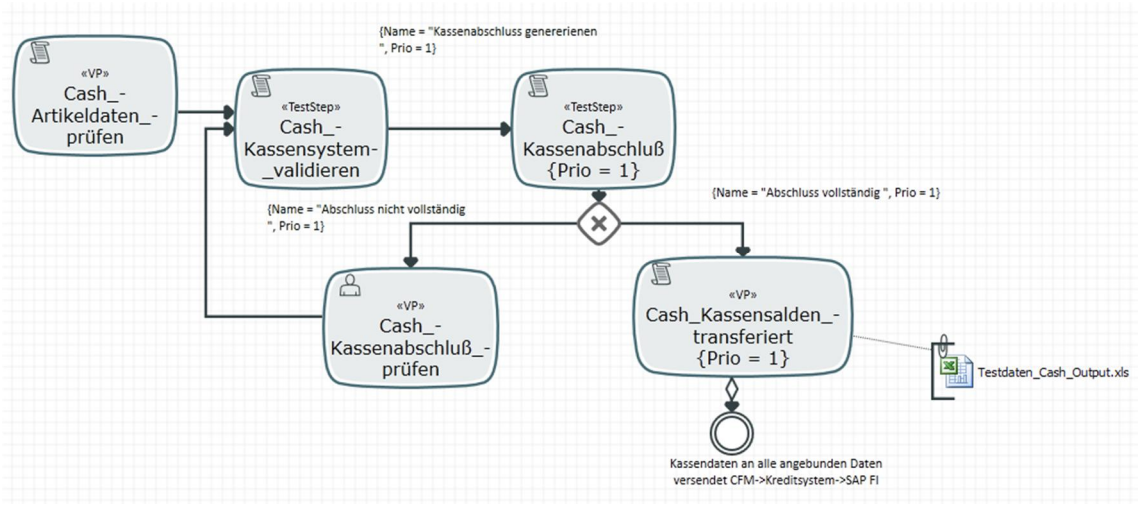


Abbildung 54: Kassenabschluss, Datentransfer zu angebundenen Softwaresystemen

Dieser Ausschnitt aus dem Testmodell „Kassensystem“ zeigt, welche Testschritte notwendig sind, um den Kassenabschluss erfolgreich durchzuführen. Ist der Kassenabschluss erfolgreich prozessiert, so generiert das System Dateien, welche an die angebundenen, weiterverarbeitenden Systeme gesendet werden. Im Rahmen der Dissertation fokussiert der Autor die Dateien, die zur Finanzbuchhaltung, zum Kreditsystem und zum CFM-System übertragen werden. Die Informationen, welche die Kasse an die Finanzbuchhaltung überträgt, sind buchhalterischer Natur. So werden sowohl die Kundennummer, der erzielte Umsatz, die offenen Salden und die Zahlungsart an SAP-FI gesendet. Für den späteren Zahlprozess sind diese Informationen von entscheidender Bedeutung. Die Spezifizierungen, welches Format und welchen Typ diese zu transferierenden Daten haben müssen, werden in der Schnittstellenspezifikation für jedes System festgelegt. Für den Test ist es wichtig, genau diese Daten abzufangen, um diese als Testgrundlage zu nutzen. Eine Alternative bilden die Fachbereiche, die die Testdaten genau kennen und gemeinsam mit einem Tester Testdaten spezifizieren können.

Nachdem ein Kunde mittels CFM angelegt und an die Kassensysteme, die Kreditsysteme und an SAP-FI übertragen worden ist, kann jetzt anhand der oben skizzierten Prozesse der Kern des IT-gestützten Zahlungsabwicklungsprozesses erläutert werden. Angenommen, ein Kunde hat Artikel und Waren eingekauft und bezahlt diese per Lastschriftverfahren (30 Tage nach Rechnungserstellung) [142], [33]. Nachdem der Kassenabschluss erfolgt ist, werden die Daten über die offenen Posten an SAP-FI übertragen. Zum jeweiligen Kunden werden die Kundennummer, der Umsatz, die Zahlungsart und das Kreditlimit übertragen. Die sich wiederholenden Aktivitäten aus den vorangegangenen Abschnitten lässt der Autor dieser Dissertation mit dem Hinweis, dass dies schon in den bereits betrachteten Unterabschnitten erläutert wurde, weg.

3.7.4 Testfallmodell ERP-System SAP-FICO – Zahl- und Mahnprozess

Der IT-gestützte Zahlungsabwicklungsprozess (Debitorenprozess) besteht neben den schon behandelten Softwaresystemen aus einem zentralen System. Das zentrale System ist die Hauptbuchhaltung, bestehend aus Hauptbuch und Nebenbüchern [142]. Die SAP bietet Unternehmen mehrere Standardsoftwarelösungen für die Verwaltung ihrer Unternehmensprozesse. Die Finanzbuchhaltung bildet in jedem Unternehmen das Herzstück der Softwaresysteme, zumindest in der Domäne der B2B-Geschäftsprozesse [142].

Die Finanzbuchhaltung besteht, unabhängig von der im jeweiligen Unternehmen eingesetzten Softwarelösung, aus mehreren Untermodulen, die beispielsweise die Verwaltung von Anlagen, das Controlling, die Abwicklung von Auftragsmodalitäten und die Zahlungsabwicklung beinhalten. Das Ziel dieser Dissertation ist es nicht, alle beteiligten Finanzmodule zu beschreiben oder in BPMN zu modellieren. Der Autor beschränkt sich für die Demonstration ausschließlich auf die im Fallbeispiel notwendigen Testprozesse zur IT-gestützten Zahlungsabwicklung. Unabhängig davon erlaubt es die hier vorgestellte Methode, alle möglichen betriebswirtschaftlichen Geschäftsprozesse sowohl für den Test zu modellieren als auch Testfälle daraus zu generieren. Der Hauptfokus liegt auf dem Zahl- und Mahnprozess sowie auf dem vom Zahlprozess ausgelösten Bankenprozess [33]. Wichtig in diesem Zusammenhang ist das Testen der Berechtigungen im SAP-FI. Deshalb wird hier dem Testschritt des Anmeldens am System und der Ausführung von SAP-Transaktionen ein eigener Unterabschnitt gewidmet. Alle hier zu demonstrierenden Funktionen und Funktionsbibliotheken, welche im Rahmen dieser Dissertation entstanden sind, lassen sich auf alle SAP-Module übertragen.

3.7.4.1 Testfallmodellierung Zahllauf „Anmelderoutine und Berechtigungsprüfung“

Das Testen von Berechtigungen in SAP-Systemen, vor allem in geschäftskritischen und sensiblen Systemen wie dem Finanzmodul, wird immer wieder vernachlässigt [144], [142]. Daher werden die Berechtigungstests im Rahmen dieser Dissertation auch einen Teil des Gesamttestmodells bilden. Die Anmelderoutine wird an dieser Stelle ausgelassen, da sie ausgiebig im CFM-Teil erläutert wurde. Die Testdaten, welche an den Testfall <<SAP_System_Transaktion_check>> geheftet sind, bestehen aus einem Benutzer mit seiner Rolle und mit allen für ihn „erlaubten“ und auch „unerlaubten“[82] Transaktionen (Negativtest). Ein Testautomat übernimmt genau an dieser Stelle das Testen der Berechtigungen für den angemeldeten Benutzer.

[82] Der Test muss dann auf einen Fehler hinauslaufen, denn Negativtests testen nicht nur die Testfälle selbst, sondern dienen auch dazu, festzustellen, ob die korrekten Berechtigungen im System vergeben sind.

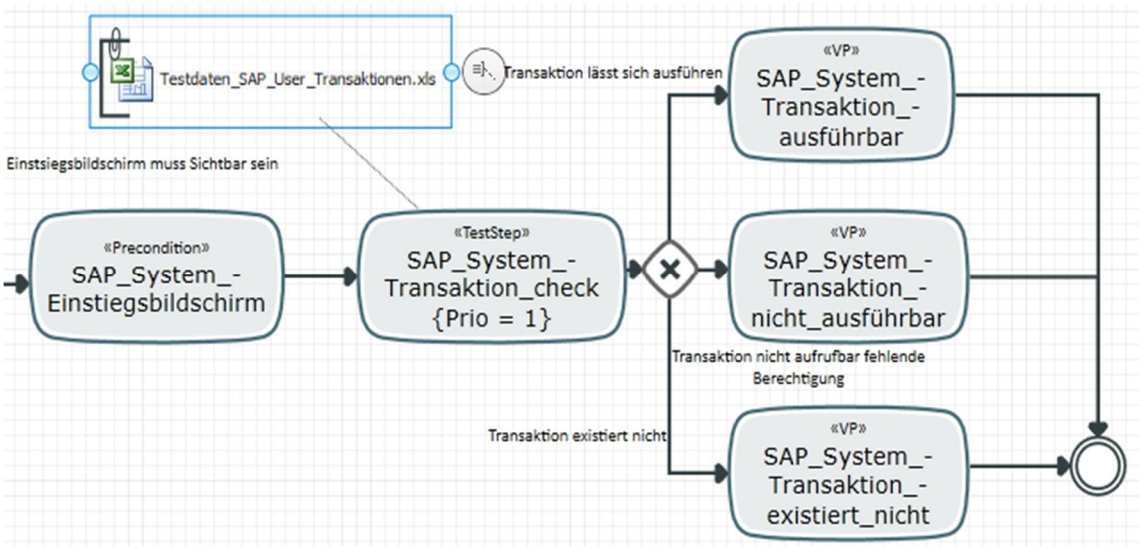


Abbildung 55: Testmodell zum Berechtigungscheck im SAP, angelehnt an [84]

Dabei können drei Arten von Fehlerwirkungen erreicht werden. Entweder die Transaktion ist passend zum aufrufenden Benutzer und wird aufgerufen oder die Transaktion kann nicht aufgerufen werden. Im Falle des Nichtaufrufs ist zu prüfen, ob dies ein bewusstes Erwirken dieses Zustands ist oder ob es sich tatsächlich um einen Fehler handelt. D. h., es könnten in den Testdaten dem Benutzer Transaktionen zugeordnet worden sein, die er tatsächlich im System nicht besitzen darf. Dann wäre das ein erfolgreicher Negativtest. Die dritte und letzte Wirkung ist, dass eine Transaktion im System nicht existiert. Das kann zwei Gründe haben: Entweder ist die Transaktion falsch eingetippt worden oder die Transaktion wurde mittlerweile im SAP-System deaktiviert oder gelöscht.

Für das Testen der Berechtigungen und der Transaktionen reicht ein generisches Testskript. Dieses übernimmt den Test von mehr als 6000[83] Transaktionen im SAP-Modul FI. Das Testmodell wirkt unterstützend bei der Spezifizierung der Varianten und der Wirkungen. Die Berechtigungsprüfung bildet den Grundstein für den darauffolgenden Test der Geschäftsprozesse. Erfahrungswerte, welche der Autor dieser Dissertation aus Interviews mit Fachbereichen und Supportabteilungen gewinnen konnte, belegen, dass hier Handlungsbedarf beim Testen besteht.

3.7.4.2 Testschritt im Testmodell: SAP-System-Transaktion setzen

Für diesen Testschritt wurde eine Funktion angefertigt, welche generisch für alle SAP-Module genutzt werden kann. Folgendes Programmstück illustriert, wie das Transaktionseingabefenster aktiviert wird und wie die Transaktionen in das Eingabefenster automatisiert eingegeben werden.

[83] www.sap.com.

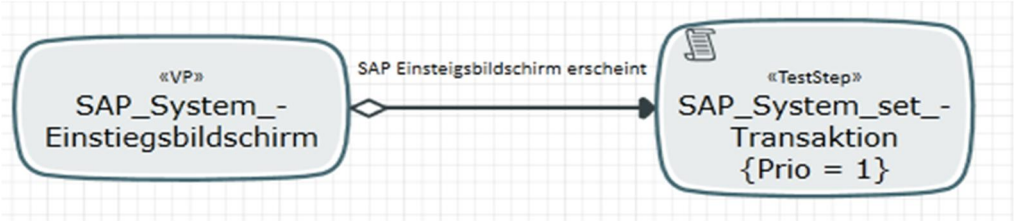


Abbildung 56: Testmodellausschnitt Transaktionspflege

Die Bedingung für das Starten des Transaktionsaufrufs ist das Vorhandensein des SAP-Einstiegsbildschirms. Dieser kann als Verifikationspunkt im Testskript mit folgender Funktion getestet werden:

```
130  function sap_initial_screen
131      SAPGuiSession(Session).SAPGuiWindow(SAPWin).SAPGuiOKCode(SAPOK).Set "/n"
132      SAPGuiSession(Session).SAPGuiWindow(SAPWin).SAPGuiButton(SAPForward).Click
133  End function
```

Diese Funktion ist generisch und allgemeingültig für jede SAP-Applikation. Mit dem SAP-Standardbefehl in Zeile 131 „/n“ wird immer der Einstiegsbildschirm nach einer erfolgreichen Anmeldung erzwungen. Im nächsten Schritt soll jetzt demonstriert werden, wie der Testfall <<SAP_System_set_Transaktion>> die jeweils für den Prozess notwendigen Transaktionen automatisiert in das Eingabefenster einträgt und ausführt.

```
135  function sap_transaction_set(psTransaction)
136      SAPGuiSession(Session).SAPGuiWindow(SAPWin).SAPGuiOKCode(SAPOK).Set psTransaction
137      SAPGuiSession(Session).SAPGuiWindow(SAPWin).SAPGuiButton(SAPForward).Click
138  End function
```

Mittels „psTransaction“ werden jeweils die aufzurufenden Transaktionen aus den Testdaten übergeben und ausgeführt. Eine zusätzliche Funktion, welche in der Lage ist, jede Fehler- oder Warnmeldung aus SAP auszulesen, ist die Funktion:

Public Function sap_status_data_res (pbExp),

welche es ermöglicht, dass alle Statusmeldungen aus der Statusbar oder aus Dialogboxen in SAP ausgelesen werden. Mit den schon vorgestellten Funktionen zum Aufzeichnen von einzelnen Testschritten bilden diese eine gute Kombination zur lückenlosen Dokumentation aller Testschritte in SAP. Nachdem die beiden Testschritte aus dem obigen Testmodellausschnitt vorgestellt worden sind, folgt jetzt der konkrete Aufruf einer Transaktion, welche einen Geschäftsprozess ausführt.

3.7.4.3 Testfallmodellierung: Zahlungsvorschlag einplanen – bearbeiten und Zahlungsvorschlagsliste

Mit der SAP-Standardtransaktion „F150“ muss jetzt ein Zahlungsvorschlag ermittelt werden [84], [33]. Dazu benötigt das SAP die von der Kasse gemeldeten offenen Posten eines Kunden mit der korrespondierenden Debitorennummer in der Buchhaltung.

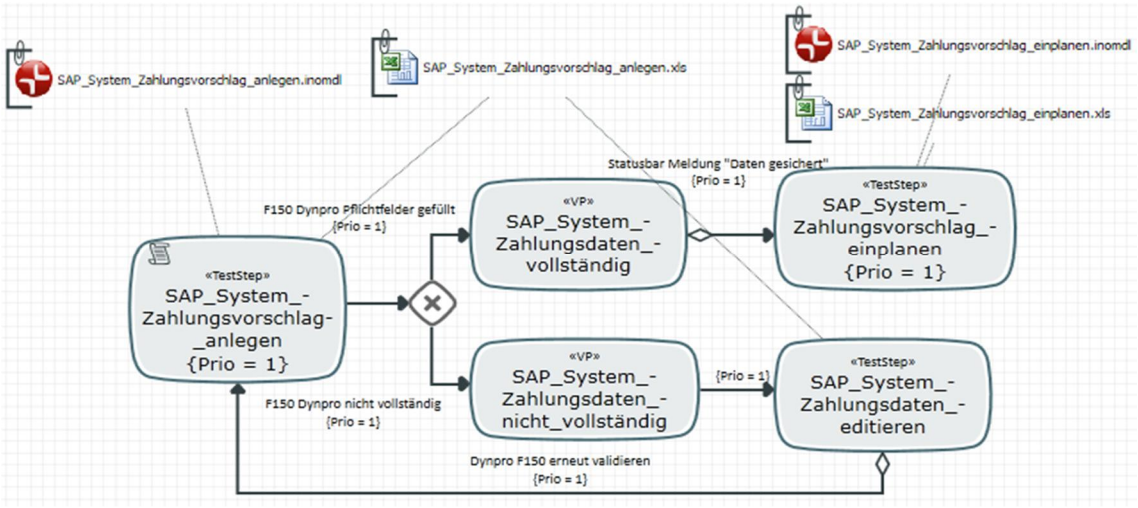


Abbildung 57: Testmodellausschnitt, Prozess angelehnt an [84]

Was hier genau passiert: Nachdem erfolgreich die Transaktion „F110“ ausgeführt wurde, muss das geladene Dynpro[84] mit Testdaten gefüllt werden [84]. Damit dieser anzulegende Testlauf von anderen im SAP-System zu unterscheiden ist, empfiehlt es sich, im Datenfeld „Identifikation“ das Tagesdatum zu wählen, gleichzeitig muss das Feld „Ausführungsdatum“ ausgefüllt werden [33], [84]. Weitere für den Test notwendige Parameter sind das Buchungsdatum und die zu erfassenden Belege (Nummernkreise). Aus diesen Belegen werden nur die Belege selektiert, welche das tagesaktuelle Buchungsdatum haben. Dies signalisiert automatisch die Fälligkeiten der einem Kunden zugeordneten Rechnungen und damit den Start des Zahlprozesses. Daraus resultiert dann die Debitorenliste. Die Debitorenliste schwankt in der Anzahl der darin enthaltenen Zahlungsanweisungen. Beim betrachteten Beispiel besteht die Liste aus 300.000 Belegen. Jeder dieser Belege signalisiert eine offene und per Lastschrift zu begleichende Rechnung eines Debitors.

Eine SAP-Regel prüft nach dem Erfassen der Testdaten im Testschritt <<SAP_System_Zahlungsvorschlag_anlegen>> durch das Klicken des „Sichern“-Buttons die erfassten Daten. Auch alle möglichen Arten von Buttons, welche SAP im Standardpaket bietet, wurden vom Autor dieser Dissertation in einer Funktionsbibliothek zusammengefasst. Die Funktion:

```
Function blnSAPButtonExist (objSAPElement)
```

fasst alle verfügbaren Elemente zusammen, dabei wurde darauf geachtet, die Buttons nach ihrem tatsächlichen Objektnamen im SAP zu benennen. Im obigen Programmstück illustrierten schon die beiden Zeilen 103 und 104, wie die Buttons „OK“, „Enter“ oder auch der Button „Vorwärts navigieren“ benannt werden [84]. Das Erfassen der Daten im SAP erfolgt automatisiert durch die Funktion: „Function SAP_System_Zahlungsvorschlag_anlegen“, welche sich in der Funktionsbibliothek befindet. Hinter dieser Funktion verbirgt sich die Standardtestlogik der Transaktion „F110“ und gesteuert werden die Abläufe durch die spezifizierten Testdaten. Sowohl die im Testmodell hinterlegten Verifikationspunkte als auch die Bedingungen werden im Testskript abgefangen. Insgesamt agieren hier folgende Funktionen miteinander:

[84] Dynpro steht in SAP für dynamische Programme, welche über Transaktionen aufgerufen werden können.

Function SAP_System_Zahlungsvorschlag_anlegen

Function SAP_System_Zahlungsvorschlag_editieren

Function SAP_System_Zahlungsvorschlag_einplanen

Beide Verifikationspunkte werden durch die Standardtestprüfroutinen, welche sich verteilt in den Funktionsbibliotheken befinden, durchgeführt. Sind diese Testschritte vollzogen, so werden beide angezeigten SAP-Dynpro-Elemente mit einem „Bestätigen“ geschlossen.

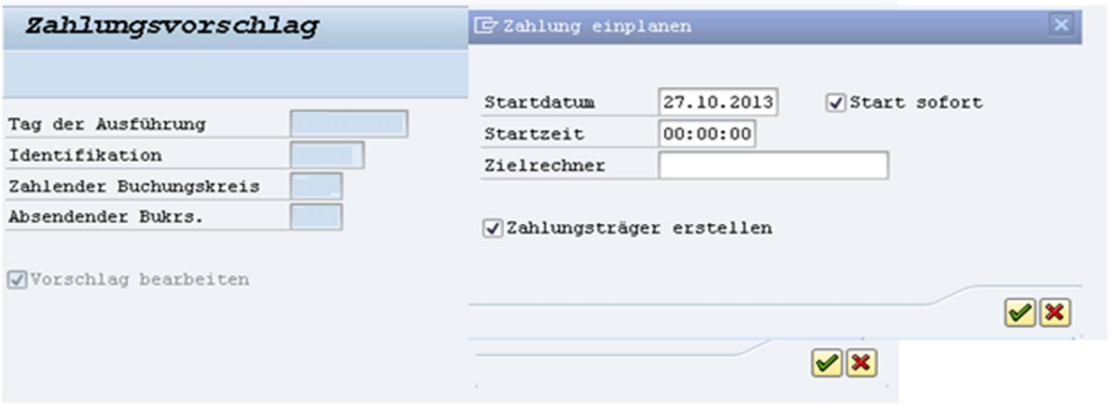


Abbildung 58: SAP-Dynpro „Zahlvorschlag anlegen“

Der folgende kleine Programmausschnitt aus der Funktion (SAP_System_Zahlungsvorschlag_anlegen), welche die Transaktion „F110-Testablauflogik“ implementiert, demonstriert, wie die in Abbildung 58 aufgeführten Felder initialisiert und mit Testdaten versehen werden. Das vollständige Testprogramm zum Abwickeln des Zahlungslaufs besitzt ca. 2000 Programmcodezeilen in VBA.

```
140  function SAP_System_Zahlungsvorschlag_anlegen
141      psTransaction = DataTable.Value("Transaktion","Initial")
142      'Aulesen der zu nutztenden Transaktion F110
143      sap_transaction_set(psTransaction)
144
145      SAPGuiSession("Session").SAPGuiWindow("Automatic Payment Transactions")
146      .SAPGuiEdit("Run Date").Set DataTable.Value ("Run_Date","Initial")
147
148          'Identification
149          hh=left(time,2)
150          mm= right(left(time,5),2)
151          s = right(left(time,7),1)
152          sid = hh&mm&s
153  SAPGuiSession("Session").SAPGuiWindow("Automatic Payment Transactions").SendKey ENTER
154
155      ...
156
157  End function
```

Wie holt sich das SAP-System jetzt die offenen Posten von einem Kassensystem? Das Einlesen der von der Kasse gemeldeten Umsätze zu einem bestimmten Debitor sowie zu seinen offenen Salden erfolgt durch den Aufruf der SAP-Transaktion „Y03_STORE“ [33]. Diese sorgt dafür, dass die Datei, welche von der Kassenschnittstelle bereitgestellt und in das SDI des SAP-Systems geladen [84] wurde, abgeholt wird. Der Testbaustein sowie das

Testmodell (nicht vollständig – nur ein kleiner Ausschnitt), das dafür verantwortlich ist, sieht wie folgt aus. Ein manueller Eingriff im Prozess ist hier notwendig, wenn die Kassendatei nicht validiert werden kann. Das Validieren übernimmt das SAP-System, indem es die Struktur der Datei prüft. Parallel dazu überprüft ein Testskript die im Excelformat erfassten Testdaten auf Vollständigkeit. Dazu werden immer wieder schon vorher implementierte Funktionsbibliotheken aus den vorangegangenen Unterabschnitten herangezogen.

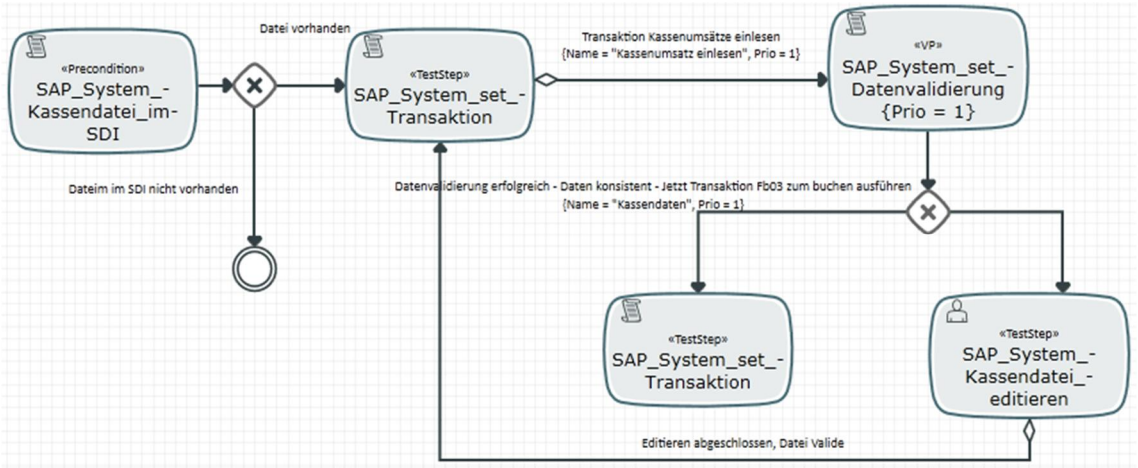


Abbildung 59: Testmodellausschnitt SAP-Kasse

Nur wenn die Validierung der Datei und die Testdatenprüfung durch das Testskript „Function SAP_System_set_Datenvalidierung“ erfolgreich sind, wird die Transaktion „Y03“ übergeben, so können die gemeldeten Kassendaten im SAP-System geladen und mit der Standardtransaktion „FB03“ verbucht werden [33]. Über diese Schnittstelle gelangen die Kassendaten in das SAP-System. Die Verbuchung der Dateien im System erfolgt wieder nach einem SAP-Standardprozess. Zunächst wird die Transaktion „YSDI“ aufgerufen, dann wird eine sogenannte „BATCH-INPUT-MAPPE“ angelegt, diese Mappe wird dann mit der SAP-Standardtransaktion „SM35“ abgespielt [33]. Das Anzeigen und Analysieren der verbuchten Kassenbelege kann dann wieder mit der SAP-Standardtransaktion „FB03“ durchgeführt werden. Der Testschritt <<SAP_System_set_Transaktion>> kommt hierbei mehrmals vor, so reicht es immer wieder, diesen einmal generisch programmierten Funktionsbaustein zu nutzen. Dieser ist verantwortlich für das Ausführen aller hier erwähnten Standardtransaktionen in SAP.

Stehen die Daten im SAP-System und sind sie erfolgreich validiert, so können sie jetzt für den anstehenden Zahllauf genutzt werden. Jetzt gilt es, konkret den Zahlprozess zu starten. Die Zahlungsvorschlagsliste ergibt sich aus den Daten der verarbeiteten Kassendatei.

3.7.4.4 Testfallmodellierung Zahlungsträger erstellen – Bankenprozess

Um den Prozess in seiner Gesamtheit darzustellen, benötigt es viele einzelne Elemente, die zu einer Überladung dieses Kernteils der Dissertation führen würden. So fokussiert der Autor die wesentlichsten und entscheidendsten Elemente. Der folgende Teilausschnitt aus dem Testmodell illustriert, welche Testschritte vom Zahllaufstart bis zum eigentlichen Einziehen der offenen Forderungen erfolgen.

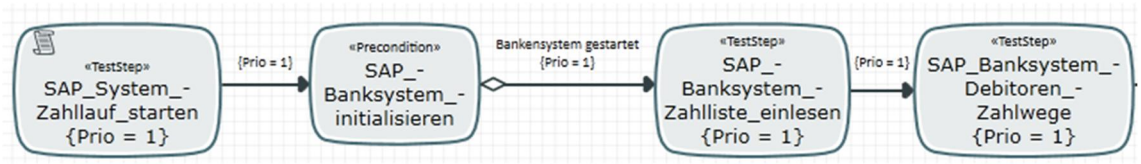


Abbildung 60: Testmodellausschnitt Zahllauf und Zahlwege ermitteln

Dieser Prozess veranschaulicht, welche Testschritte hier vonnöten sind. Die Funktionen

Function SAP_System_Zahllauf_starten

Function SAP_Banksystem_initialisieren

Function SAP_SAP_Banksystem_Zahlliste_einlesen

Function SAP_Banksystem_Debitoren_zahlwege

werden alle durch die Funktion SAP_System_set_Transaktionen aufgerufen. Mit der Transaktion F110 wird jetzt der Zahllauf gestartet [84]. Diese Transaktion impliziert eine weitere Transaktion, welche das SAP-Bankenmodul initialisiert, das dafür verantwortlich ist, dass die Lastschriften auf die Unternehmenshausbanken gutgeschrieben werden. Die von den Kassen ermittelten Belege, welche eingezogen werden sollen, befinden sich dann in der Zahlliste.

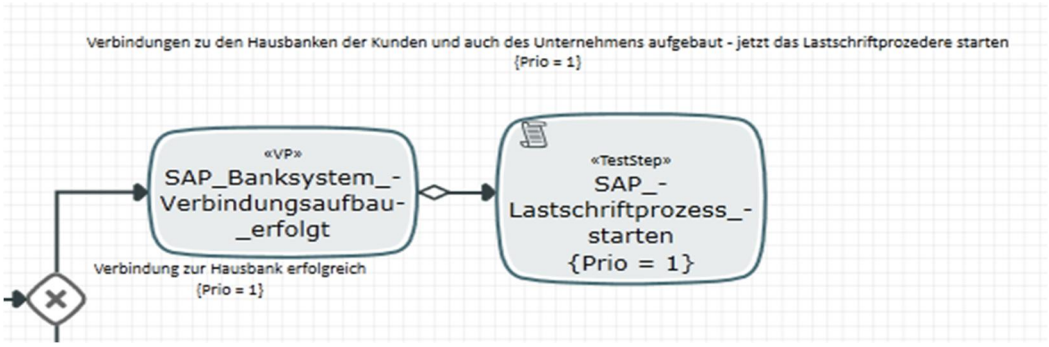


Abbildung 61: Teilmodellausschnitt Bankensystem Verbindungsaufbau

Durch SAP-Prüfroutinen, welche auch im Testskript abgefangen werden, können Verbindungsaufbauten zu Fremdsystemen überprüft werden. Die Bedingung, welche im Testmodell dargestellt ist, soll demonstrieren, dass nur bei erfolgreichem Verbinden mit den Bankensystemen der Lastschriftprozess gestartet werden soll. Mit dem Testschritt <<SAP_Lastschriftprozess_starten>> werden ca. 300.000 Belege, welche sich in der Zahlliste befinden, sukzessive abgearbeitet. Nach der Abarbeitung kann es drei mögliche Zustände geben.

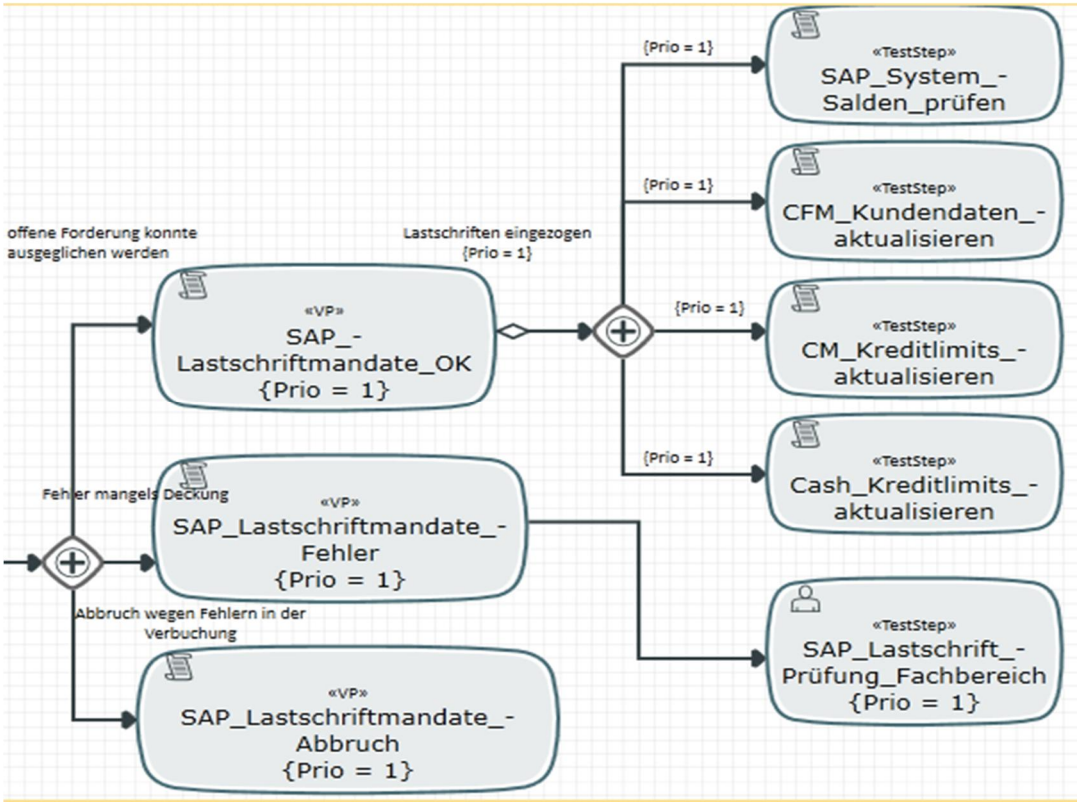


Abbildung 62: Testmodellausschnitt: Lastschriftprozess

Insgesamt können drei Dateien nach dem Lastschriftprozess generiert werden. Eine Datei enthält eine Liste aller Debitoren, bei denen der Lastschriftprozess abgebrochen wurde, eine Datei enthält eine Liste aller Debitoren, bei denen der Lastschriftprozess aufgrund von Fehlern abgebrochen wurde. Und es gibt eine Datei, welche alle erfolgreich durchgeführten Lastschriftmandate enthält. Bei Fehlern kann es sein, dass bestimmte Lastschriftmandate trotz Datenvalidierung Fehler enthalten können. Typische Fehler sind „Zahlendreher“ bei den Kontoinformationen oder auch ganz einfache Tippfehler. Der Zahlprozess endet immer mit der Generierung dieser drei Dateien und mit einem Gesamtprotokoll.

In einem weiteren Schritt müssen jetzt die Ergebnisse aus dem Zahllauf verarbeitet werden. So muss im Falle eines Abbruchs der Transaktion im Lastschriftprozess aufgrund von Nichtdeckung des Kontos der Mahnprozess automatisch ausgelöst werden. Aber bevor dieser Schritt erfolgt, müssen mit dem Testschritt <<SAP_System_Salden_prüfen>> und mit der Transaktion „FB5LN“ die gebuchten Debitoren zu den Personenkonten überprüft werden [84], [33].

```
160 Function SAP_System_Salden_prüfen(vendor, CompanyCode, Document)
161     'Testdaten einlesen
162     qc_ImportSheet psPath, psFile, "Salden_prüfen", "Salden_prüfen"
163     psTransaction = DataTable.Value("Salden_prüfen", "Transaktion")
164         mgi_sap_transaction_set psTransaction
165         With .SAPGUISession("Session")
166         .SAPGuiWindow("Customer Line Item Display").SAPGuiEdit("Customer account").Set vendor
167         .SAPGuiWindow("Customer Line Item Display").SAPGuiEdit("Company code").Set CompanyCode
168         .SAPGuiWindow("Customer Line Item Display").SAPGuiButton("Dynamic selections   (Shift+F4)").Click
169         .SAPGuiWindow("Customer Line Item Display").SAPGuiEdit("Document Number").Set Document
170         .SAPGuiWindow("Customer Line Item Display").SAPGuiEdit("Open at key date").Set ""
171         .SAPGuiWindow("Customer Line Item Display").SAPGuiRadioButton("Open items").Set
172         .SAPGuiWindow("Customer Line Item Display").SAPGuiCheckBox("Normal items").Set "ON"
173         .SAPGuiWindow("Customer Line Item Display").SAPGuiCheckBox("Special G/L transactions").Set "ON"
174         .SAPGuiWindow("Customer Line Item Display").SAPGuiCheckBox("Noted items").Set "ON"
175         .SAPGuiWindow("Customer Line Item Display").SAPGuiCheckBox("Parked items").Set "ON"
176         .SAPGuiWindow("Customer Line Item Display").SAPGuiCheckBox("Vendor items").Set "ON"
177         .SAPGuiWindow("Customer Line Item Display").SAPGuiButton("Execute   (F8)").Click
178         If .SAPGuiWindow("Customer Line Item Display_2").SAPGuiStatusBar("StatusBar").Exist Then
179             If .SAPGuiWindow("Customer Line Item Display_2").SAPGuiStatusBar("StatusBar").GetROProperty("messagetype")="S"
180                 Reporter.ReportEvent micPass, "Open Items", "Folgende offene Posten sind vorhanden: "&Document
181                 capture_public SAPGuiSession("Session").SAPGuiWindow("Customer Line Item Display_2"), "Display"
182             End If
183         Else
184             Reporter.ReportEvent micFail, "Open Items", "Es sind keine offene Posten vorhanden."
185             capture_public SAPGuiSession("Session").SAPGuiWindow("Customer Line Item Display_2"), "Display"
186         End If
187         SapGuiUtil.CloseConnections
188         End With
189 End Function
```

Dieses Programmstück selektiert im SAP-System die gebuchten und die nicht gebuchten Posten und vergleicht diese mit den beim Zahlprozess generierten Dateien. Die Transaktion, welche hier im Programmausschnitt demonstriert wird, prüft auch anhand von Statusmeldungen in der Statusbar den Erfolg oder Misserfolg der Verarbeitung (Zeilen 179–185). Wobei die Zeilen 181 und 185 gleichzeitig dafür sorgen, dass zu den angezeigten Belegen Aufzeichnungen gemacht werden, welche die Testauswertungen später erleichtern. Sind die Zahlungen korrekt auf die jeweiligen Unternehmenskonten (Banken) verbucht, so können mit der Transaktion „YFI_SAPF00“ [33] die Debitorenkonten ausgeglichen werden. Diese Transaktion löst automatisch im SAP weitere Prozesse aus, wie bspw. das automatisierte Versenden der Daten an abhängige Softwaresysteme wie CFM, an Kreditsysteme und an die Kassensysteme. Durch die Modellierung der Testschritte <<CFM_Kundendaten_aktualisieren>>, <<Cash_Kreditlimit_aktualisieren>> und <<CM_Kreditlimits_aktualisieren>> werden die Testprozesse in den angebundenen Systemen wieder gestartet, um zu überprüfen, ob die von SAP übertragenen Daten auch tatsächlich in den Systemen verarbeitet wurden.

Eine weitere kritische Schnittstelle ist das Zurücksetzen der Kreditlimits [33]. Hat ein Kunde per Rechnung und auf Lastschrift eingekauft, so gewährt ihm das Unternehmen einen Kredit. Die Kredithöhe, die der Kunde maximal erhalten darf, ist ja eingangs durch das Kreditsystem ermittelt worden. Angenommen, der Kunde hat bspw. ein Kreditlimit von 10.000 Euro. Kauft er am 1.1.2014 für 4000 Euro auf Rechnung, so reduziert sich das Kreditlimit nach dem 1.1.2014 auf 6000 Euro. Begleicht er die Rechnung fristgerecht durch das erfolgreiche Durchführen des Zahlprozesses im SAP, so muss sein Kreditlimit wieder auf den Ursprungswert von 10.000 Euro gesetzt werden. Kommt der Kunde in Zahlungsverzug, bspw. wenn die Lastschriftmandate nicht erfolgreich eingezogen werden konnten, müssen Mahnprozesse folgen. Ein dreistufiges Mahnverfahren, welches angewandt wird [142], tritt im Testschritt <<SAP_Lastschriftmandate_Abbruch>> ein.

Im Testschritt <<SAP_Lastschriftmandate_Fehler>> kommt es zwar auch zum Abbruch, aber die Ursache der Abbrüche könnten bspw. „Zahlendreher“ bei den Kontodaten

sein usw. Hier muss der Fachbereich manuell eingreifen, um diese Fehler zu korrigieren. Bei der SAP-Fehlermeldung <<SAP_Lastschriftmandate_Abbruch>> handelt es sich um nicht gedeckte Konten, sprich der offene Saldo konnte nicht ausgeglichen werden [84]. Hier läuft der SAP-Standardprozess weiter, und zwar mit dem Starten der Mahnprozesse. Gleichzeitig bleibt im ersten Mahnschritt das Kreditlimit des Kunden unberührt.

3.7.4.5 Testfallmodellierung Mahnlauf

Der Mahnlaufprozess erfolgt analog zum Zahllaufprozess. Daher werden hier Elemente, welche analog zum Zahlprozess [142], [33] erfolgen, nicht näher betrachtet, sondern nur Elemente, welche bisher im Zahllaufprozess nicht enthalten waren. Auch hier muss ein Mahnvorschlag in Form einer Mahnliste durchgeführt werden. Die Mahnliste wird automatisch durch das SAP-System erstellt und enthält Belege, welche der abgebrochene Zahlprozess generiert [33]. Hinter jedem Zahlungsbeleg verbirgt sich ein Kunde. Der Mahnprozess ist ein Standard-SAP-Prozess, welcher mit Unternehmensrichtlinien angereichert wird. Ein dreistufiges Mahnverfahren ist auch in der Praxiswelt das standardisierte Vorgehen vieler Unternehmen [142].

Hinter jeder Mahnstufe stehen Verifikationsaktivitäten, welche zu einem festen Zeitpunkt durchgeführt werden müssen. Dazu gehört auch der oben dargestellte Zahlprozess. Dieser wird nach dem ersten Mahnverfahren erneut ausgelöst, mit dem Versuch, die Rücklastschrift einzulösen.

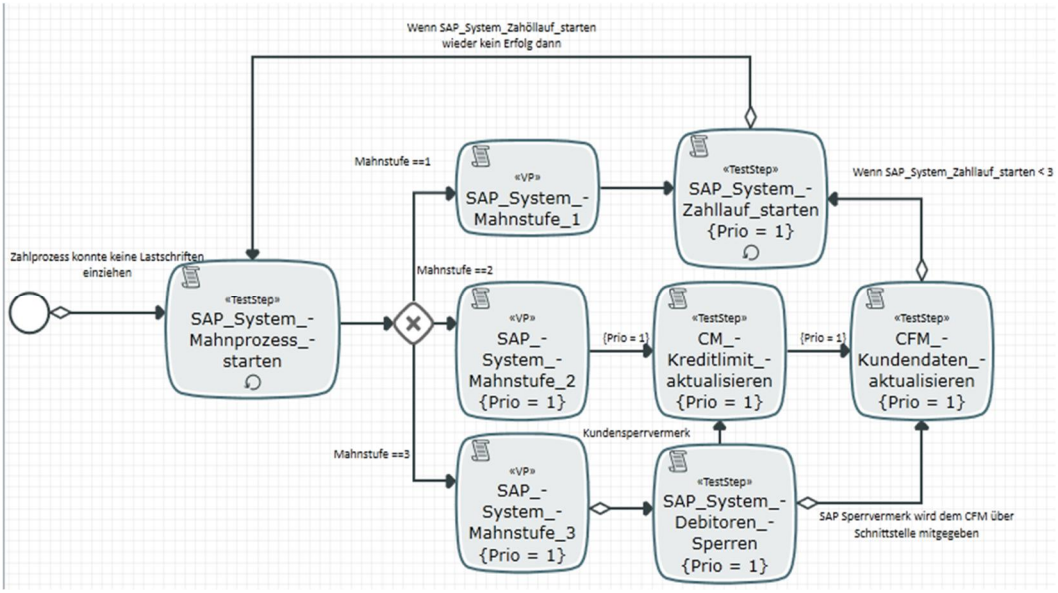


Abbildung 63: Mahnprozess im SAP-System, angelehnt an [33] und [142]

Erst wenn dies auch keinen Erfolg bringt, werden die weiteren Mahnstufen je nach Bedingung ausgeführt. Die Testfallgenerierung kann an dieser Stelle nachträglich umsortiert werden. Wichtig ist, dass der komplette Mahnprozess, sprich alle drei Mahnstufen, während eines Regressionstests getestet werden muss.

Analog zum Zahlprozess existieren auch für den Mahnprozess mit den modellierten Testschritten korrespondierende Funktionsbibliotheken. Zur Verdeutlichung reicht es im Rahmen dieser Dissertation aus, den ersten Testschritt <<SAP_System_Mahnprozess_starten>> zu demonstrieren.

Dieses Programmstück startet den Mahnprozess und nutzt die zuvor vom Zahlprozess

```
function SAP_System_Mahnprozess_starten
    psTransaction = DataTable.Value("Transaktion","Initial")
    'Aulesen der zu nutztenden Transaktion F150
    sap_transaction_set(psTransaction)
    With

        .SAPGuiWindow("Dunning").SAPGuiEdit("Run On").Set DataTable.Value("Run_On","Initial")
        .SAPGuiWindow("Dunning").SAPGuiEdit("Identification").Set sid
        .SAPGuiWindow("Dunning").SAPGuiTabStrip("DU_TABSTRIP").Select "Parameter"
        .SAPGuiWindow("Dunning").SAPGuiEdit("Dunning date").Set DataTable.Value("Dunning_date","Initial")
        .SAPGuiWindow("Dunning").SAPGuiEdit("Docmnts posted up to").Set DataTable.Value("Docs_posted_up_to","Initial")
        .SAPGuiWindow("Dunning").SAPGuiEdit("Company Code").Set DataTable.Value("Company_code","Initial")
        .SAPGuiWindow("Dunning").SAPGuiEdit("Customer").Set DataTable.Value("Customer","Initial")
        .SAPGuiWindow("Dunning").SAPGuiEdit("Vendor").Set DataTable.Value("Vendor","Initial")
        .SAPGuiWindow("Dunning").SAPGuiTabStrip("DU_TABSTRIP").Select "Free selection"

        If ("Session").SAPGuiWindow("Dunning").SAPGuiStatusBar("StatusBar").GetROProperty("messagetype") ="E" Then
               Reporter.ReportEvent micFail, "Fehler - Dunning List", ("Session").SAPGuiWindow("Dunning").SAPGuiStatusBar
               mgi_capture_public ("Session").SAPGuiWindow("Dunning"), "Dunning List"
               Exit
        End If

        ...

        End function
```

angelegte Datei der fehlgeschlagenen Lastschriftmandate aufgrund der Nichtdeckung des Kundenkontos. Im Anschluss an den buchhalterischen Mahnprozess erfolgen der automatische „Druckauftrag“ und der „Versende“-Prozess, welcher nicht Gegenstand dieser Dissertation ist.

Viele der hier betrachteten Testschritte und Funktionen sind erst durch diese Dissertation entstanden, so dass jetzt zum Ende dieses Kapitels eine Sammlung kleiner Funktionsbibliotheken entstanden ist, welche es erlaubt, auf der Basis von zuvor angefertigten Testmodellen und generierten manuellen Testfällen fertig automatisierte Testskripte zu erhalten. Diese können sukzessive durch weitere Softwaresysteme erweitert werden. Auch konnten zuvor erstellte Funktionen wiederverwendet und in die neuen Funktionen integriert und überführt werden. Die hier implementierten Funktionen sind so modularisiert und strukturiert aufgebaut, dass sie als Prozesskette nach einem Baukastenprinzip zusammengefügt werden können [76], [154].

Ein Erfolgsfaktor dieser Methode ist die Einfachheit, mit welcher Fachbereiche und Tester an Testskripte kommen können. Daher ist es umso wichtiger, strukturierte Programme zu haben, welche eine hohe Wartbar- und Erweiterbarkeit aufweisen. Mit Hilfe von Testmodellen, welche die zu testende Software spezifizieren, und mit der skriptbasierten Testautomatisierungstechnologie ist ein Ansatz ausgearbeitet worden, mit dem auf Basis von Testmodellen und VBA-Testskripten beliebige verteilte BIS-Geschäftsprozesse End-to-End getestet werden können.

3.7.5 Testskript-Ausführung am Beispiel von CFM im HPQC

Die aus den Testmodellen generierten Testskripte, welche gleichzeitig adaptiert worden sind, werden jetzt im HPQC für das Ausführen präpariert.

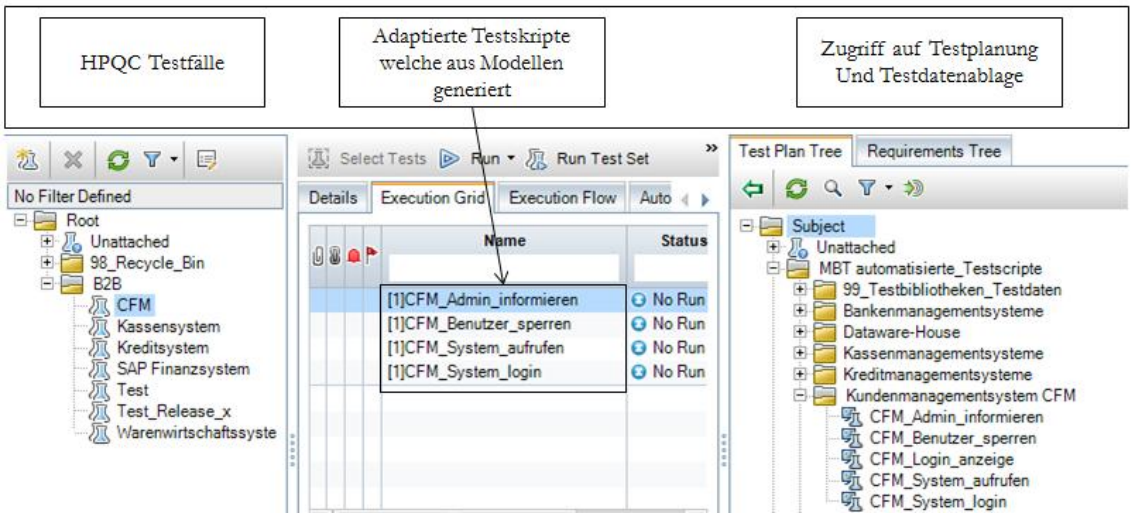


Abbildung 64: Testlauf starten aus Testsuite

Die Abbildung zeigt, wie die generierten und adaptierten Testskripte im Testscheduler für den Testlauf vorbereitet werden. Im rechten Teil der Abbildung kann zwischen der Baumansicht der generierten Testfälle und den zugehörigen Testanforderungen im Requirementmodul navigiert werden. Der Testlauf kann durch das „Klicken“ auf den „Run Test Set“-Button initialisiert und gestartet werden.

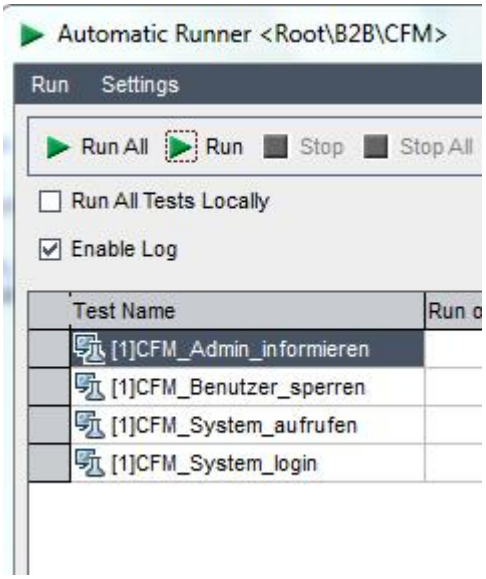


Abbildung 65: Testscheduler implizit aus Testsuite gestartet

Die Testausführungseinheit übernimmt die Abarbeitung der Testfälle und gleichzeitig den Aufruf der auszuführenden Testskripte aus QTP [161]. Die Ergebnisse der Testläufe werden direkt in die Testauswertungsformulare von HPQC und QTP geschrieben und stehen somit den Testauswertern zur Verfügung. Die auszuführenden Testskripte verkörpern kleine Programme, die wiederum Programme testen. Diese Programme müssen einem automatischen Wartungsprozess unterzogen werden, unabhängig von der Generierung von Testfällen aus Testmodellen.

3.7.6 Programmierung von wartbaren Testskripten

Die Wartbarkeit bei automatisierten Tests muss einen höheren Stellenwert erhalten, als es bei manuellen Tests der Fall ist. Hauptsächlich scheitern 60% der Testautomationsprojekte [77], [161] daran, dass die entwickelten Testskripte von vornherein nicht wartbar programmiert sind oder dass die Testskripte nicht einem ständigen Wartungszyklus unterliegen. Beide Gründe haben einen großen Anteil daran, dass viele Testautomatisierungsprojekte nicht erfolgreich sind. Die modellbasierte Testmethode, welche im Rahmen dieser Dissertation ausgearbeitet wurde, ist zwingend darauf angewiesen, dass die Testskripte funktionieren. Jedes Mal, wenn ein Testmodell angepasst werden muss, ist es notwendig, zu überprüfen, ob dies Auswirkungen auf die implementierten Funktionen hat [74]. Wenn Testschritte hinzukommen, so sorgen die entwickelten Adaptierungen dafür, dass ein Hinweis ausgegeben werden muss, der besagt, dass die Funktion keine Funktion (Testlogik) in der Funktionsbibliothek enthält.

3.7.7 Testabdeckungsgrad mit der MBT-Methode

Die Anzahl der am Prozess beteiligten Softwaresysteme bei verteilten BIS und die hohe Frequenz bei der Anzahl der im Jahr durchzuführenden Releases machten ein Umdenken beim Thema Testen in der Domäne der verteilten BIS notwendig. Im Rahmen dieser Dissertation wurde eine Methode vorgeschlagen, welche es erlaubt, Testfälle in fachlichen BPMN-Testmodellen zu entwerfen. Die Generierung der Testfälle und die anschließende Testautomatisierung sollen sukzessive sowohl die Testabdeckung erhöhen als auch die Tests beschleunigen. Die Testfälle werden über die GUI automatisiert, da es in erster Linie Kundenabnahmetests sind, welche hier modelliert werden. Durch den Aufbau von Funktionsbibliotheken konnten wiederverwendbare Funktionsbausteine entwickelt werden.

Abhängigkeiten in den Testfällen werden in Testmodellen erfasst und mit Hilfe von Verifikationspunkten und Vorbedingungen abgewickelt. Testdaten können durch die Auslagerung übergeben werden. Insgesamt konnten 230 Funktionsbausteine implementiert werden, welche von 130 Testskripten genutzt werden. Für das hier betrachtete Fallbeispiel (IT-gestützter Zahlungsabwicklungsprozess) erhöht diese Methode den Testabdeckungsgrad um 100%. Für weitere Tests müssen weitere Geschäftsprozesse in den Funktionsbibliotheken sukzessive erweitert werden. Die 100%ige Testabdeckung bezieht sich ausschließlich auf den Fall, dass ein IT-gestützter Zahlungsabwicklungsprozess per Lastschrift durchgeführt wird. Alle anderen Geschäftsprozesse sind davon zunächst einmal unberührt.

3.7.8 Zusammenfassung

Bei der Testfallspezifikation muss immer ein Fachbereich hinzugezogen werden, der dann Unklarheiten oder auch Fehler in der Modellierung sofort identifizieren kann, so können diese auch sofort korrigiert werden. Bei der Modellierung muss gleichzeitig darauf geachtet werden, dass nur die wichtigsten Aspekte und Elemente Bestandteil des Modells sind. Somit bleiben die Modelle übersichtlich und leicht lesbar.

In diesem Kapitel wurden die Problembeschreibung und der Lösungsansatz, mit welchem sich diese Dissertation auseinandersetzt, erläutert. Am Fallbeispiel der IT-gestützten Zah-

lungsabwicklung wurden der Debitoren-Prozess sowie die beteiligten Softwaresysteme vorgestellt. Dieser Prozess gilt gleichzeitig als „Best Practice“-Ansatz sowohl bei der Vorgehensweise beim Testen als auch bei der Software-Infrastruktur [158]. So wurde verdeutlicht, warum klassische Testansätze [1], [10], [119] bei komplexen, verteilten, betrieblichen Informationssystemen an ihre Grenzen stoßen und warum MBT hier eine bessere Alternative und eine Ergänzung zum klassischen Testen darstellt.

Das Kapitel befasst sich auch mit dem Thema Adaptierungen und wie mit Hilfe von programmierten Funktionsbausteinen in VBA (Visual Basic) die Lücke bisheriger MBT-Methoden[85] [30], [141] umgangen werden kann. Adaptierungen sollen dazu dienen, auf Basis von generierten manuellen (textuellen) Testfällen automatisierte und fertig ausführbare Testskripte zu adaptieren. So besteht die hier vorgeschlagene Methode aus einer Mischung des modellbasierten Testens, gepaart mit der skriptbasierten Testautomatisierung. Genau diese Mischung macht die Methode flexibel gegenüber Erweiterungen.

Basierend darauf wurden ein Lösungsansatz und die für den Lösungsansatz benötigten Voraussetzungen erläutert. Sowohl das Ergebnis der Auswahl eines Testwerkzeugs als auch die Zusammensetzung der Testwerkzeuge wurden skizziert.

[85] Die im Rahmen dieser Dissertation untersuchten MBT-Testwerkzeuge sind in der Lage, Testfälle aus Modellen wie UML bzw. BPMN zu generieren. Der Schritt zum fertig ausführbaren Testskript ist mit keinem dieser Testwerkzeuge möglich. Dazu führte der Autor dieser Dissertation Interviews und Gespräche mit der Firma SeppMed, welche das Tool MBTSuite einsetzt. Die MBTSuite ist auch im Rahmen dieser Dissertation das für die Demonstration gewählte Tool, auch in der Firma Imbus AG, vertreten durch Hrn. Roßner. Das Tool Tedeso generiert aus Testfällen Testskriptrumpfe, aber keine ausführbaren Testskripte. Gleiches gilt für Qtronic von der Firma Conformiq Tool.

4 Eingeschlagener Realisierungsweg

Dieses Kapitel bildet gemeinsam mit Kapitel 3 den Kernteil dieser Dissertation. Sowohl die technische Umsetzung der MBT-Methode als auch die Vorstellung der am Prozess beteiligten Ausschnitte von Testmodellen waren Gegenstand des Kapitels 3. Dieses Kapitel beschreibt nun den eingeschlagenen Realisierungsweg der Dissertation und erläutert sowohl die an der Lösung beteiligten Komponenten als auch die Integration.

Nachdem die Integration sowohl der Testwerkzeuge als auch des Modellierungswerkzeuges erfolgt, werden in einem weiteren Schritt die Testfallspezifikationen erstellt. Wobei die Testfallbeschreibung im Rahmen dieser Dissertation als eine einmalige „Aktion“ zu verstehen ist, welche das Ziel verfolgt, den Initialaufwand bei der ersten Testfallmodellierung komplexer Geschäftsprozesse zu minimieren. Gleichzeitig soll der einmalige Initialaufwand generell in das Thema „modellbasiertes Testen von verteilten BIS und Geschäftsprozessen“ einführen und dem Modellierer die konkrete Modellierung von komplexen Geschäftsprozessen erleichtern. Anstelle der einzelnen Testfallbeschreibungen erfolgt später nur das Testdesign, wie schon demonstriert, mittels Modellen. Wichtig ist, dass diese Testmodelle aus Anwendersicht modelliert werden und nicht auf bestehenden Entwicklungsmodellen basieren.

4.1 Skizzierung der am Lösungsansatz beteiligten Komponenten

Am einfachsten lassen sich die beteiligten Testwerkzeuge, welche gleichzeitig das für diese Dissertation zu nutzende Testframework bilden, anhand einer Grafik erläutern. In diesem Zusammenhang werden die Testwerkzeuge (Testmanagementtool, Testfallgenerator, Testautomatisierung) und das Modellierungswerkzeug gezeigt. Auf der Basis dieses Testframeworks lassen sich beliebige betriebswirtschaftliche End-to-End-Prozesse modellbasiert testen. Die rot umrandeten Aktivitäten in der Abbildung zeigen die Aktivitäten, die durch diese Dissertation hinzukommen. Die Schnittstelle zwischen dem HPQC und dem Testfallgenerator MBTSuite ermöglicht es, dass manuelle Testfälle direkt ins HPQC übertragen werden. Durch die Erweiterung der BPMN-Spezifikation mit testspezifischen Elementen sollen die aus Testersicht notwendigen Testinformationen sowie die Teststrategien festgelegt werden können. Dies ist notwendig, da eine vollständige Kombinatorik aller möglichen Testpfade aus Testmodellen zu einer Testfallexplosion führen kann.

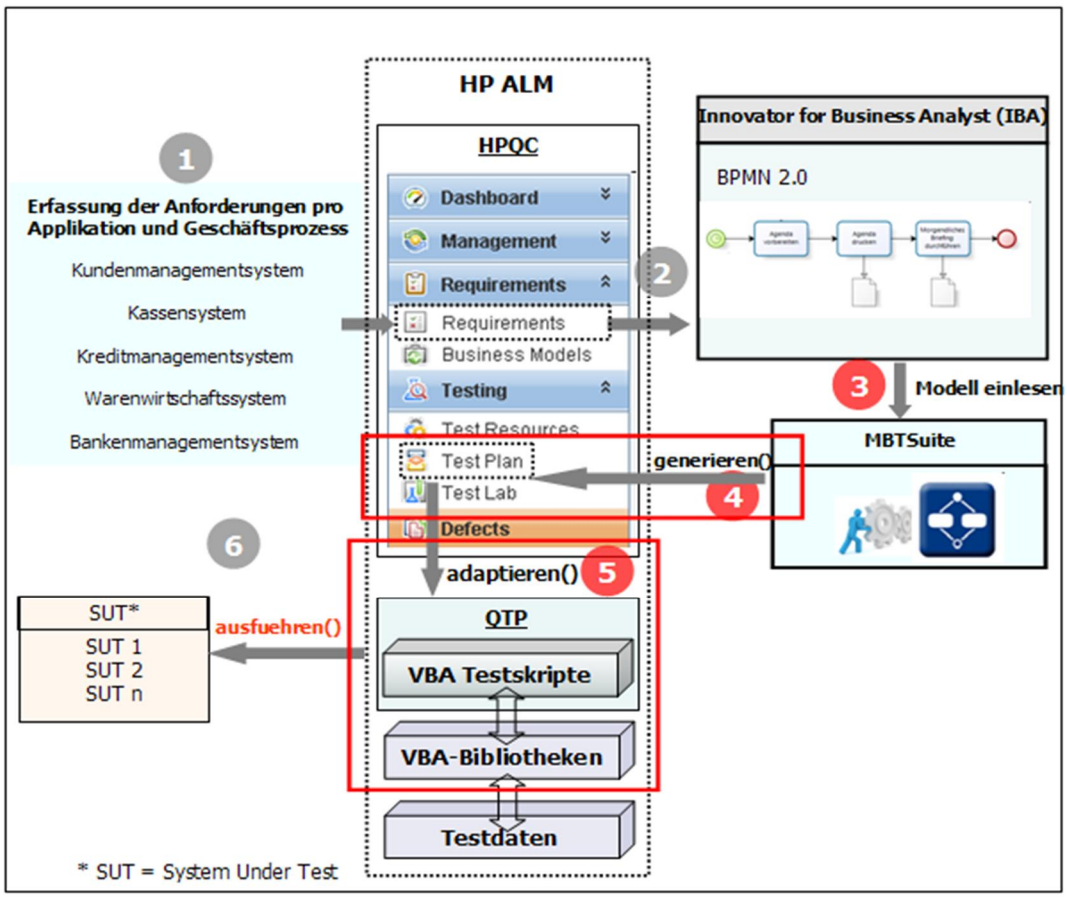


Abbildung 66: Testframework-MBT für verteilte BIS

Auf die grundlegenden Elemente reduziert, besteht ein Testmodell aus Testschritten (Teststeps), die Interaktionen mit dem zu testenden System darstellen. Zusätzlich werden Verifikationspunkte eingeführt, welche die Aufgabe haben, als Prüfinstanz zu agieren. Verifikationspunkte können an beliebigen Stellen im Testmodell hinterlegt werden.

Die Adaptierung der generierten Testskripte, welche Testskriptrumpfe darstellen, ist eine weitere Komponente, mit der sich diese Dissertation auseinandersetzt. Dargestellt wurde, wie durch Adaptierungen aus Testmodellen und generierten Testfällen fertig automatisierte und ausführbare Testskripte entstehen können.

- HPQC – Testmanagementtool, welches alle Tests verwaltet und dokumentiert. Sowohl die Testfallordnerstruktur als auch eine Namenskonvention helfen dabei, Tests zu strukturieren, und bilden gleichzeitig den Grundstein für die Automatisierung generierter manueller Testfälle.
- QTP – Entwicklungsumgebung für Testskripte und Funktionsbibliotheken. Ist für die Ausführung automatisierter Tests verantwortlich. Der Autor dieser Dissertation wählt bei der Erstellung der Testskripte verschiedene Skripttechnologien.[86] Es existieren einerseits testdatengetriebene Technologien, welche hier hauptsächlich zum Einsatz kommen, in Kombination mit schlüsselgetriebenen Skripttechnologien, welche dann zum Einsatz kommen, wenn genügend Komponenten durch Capture & Replay in der QTP Repository erfasst worden sind. Mit Hilfe von Software-

[86] Die Skripttechnologien sind sehr weit gefächert und würden hier den Rahmen sprengen, sollte der Autor auf jedes Verfahren eingehen. Eine gute Einführung in die verschiedenen Skripttechnologien bietet [9] oder aber auch das Handbuch des QTP-Testwerkzeugs von HP: HP QTP Install Guide Version 11.00.

schlüsselwörtern (wie SAP oder CFM) können dann Befehle definiert werden, welche es erlauben, auf die gesamte Funktionalität der zu testenden Software zuzugreifen. Bestes Beispiel ist das Identifizieren von Buttons oder Menüfeldern per Schlüsselwort. Die aktionsgetriebene Skripttechnologie kommt insbesondere bei der Adaptierung von Testfällen an Testskripte zum Einsatz.

- IBA – ist ein Modellierungswerkzeug, welches es erlaubt, sowohl in BPMN als auch in UML Prozessmodelle anzufertigen. Die Nutzung von IBA als Modellierungstool setzt voraus, dass integrierbare Testfallgeneratoren existieren, die mit IBA kompatibel sind, wie bspw. die MBTSuite.
- Das Produkt Excel von Microsoft dient als Testdatenbehälter – Excel wird gemeinsam mit UML-Klassendiagrammen im Rahmen des Testframeworks für die Testdatenhaltung eingesetzt. Es erlaubt Fachbereichen, Testdaten einfach in Form von Tabellen zu erfassen und diese in der HPQC-Ablagestruktur abzulegen. UML-Klassendiagramme unterstützen im Testmodell bei der Spezifizierung der Testdaten, welche dann automatisch nach Excel konvertiert/transformiert werden. Die dort hinterlegten Testdaten werden automatisch vom Testautomatisierungswerkzeug mit Hilfe zuvor implementierter Funktionen eingelesen und verarbeitet.
- Die MBTSuite ist als Testfallgenerator manueller Testfälle bekannt und übernimmt gleichzeitig die Verantwortung für die Übertragung der Testfälle ins HPQC. Bislang wird die MBTSuite sowohl in der Medizin- als auch in der Automobilindustrie insbesondere bei eingebetteten Softwaresystemen zur Qualitätssicherung eingesetzt. Dort werden bspw. aus Zustandsdiagrammen mit Hilfe von Algorithmen Testfälle generiert.

Die Adaptierung demonstrierte gemeinsam mit der BPMN-Erweiterung, wie aus den mittels BPMN gewonnenen manuellen Testfällen, welche einzelne Testschritte beschreiben, fertig ausführbare Testskripte angefertigt werden.

Ausgangspunkt sind die Testfallspezifikationen der zu testenden Softwaresysteme [127]. Auf Basis dieser Testfallspezifikationen erfolgt die Modellierung der Testfälle mit IBA. Die Modellierung und die Generierung von Testfällen erfolgen mit getrennten Werkzeugen. Die MBTSuite exportiert die mittels IBA angefertigten Testmodelle und generiert daraus manuelle Testfälle (textuell). Damit von vornherein Testfälle so aufbereitet werden, dass diese für sich „sprechend“ sind, müssen Namenskonventionen festgelegt werden. Da Namenskonventionen alle Testfälle betreffen, müssen sie deshalb schon im Testmodell berücksichtigt werden. Gleichzeitig sind Namenskonventionen auch für die Adaptierung der manuellen Testfälle an automatisierte Testskripte wichtig. Hinter Adaptierungen verbergen sich VBA-Programmstücke, die dafür verantwortlich sind, die generierten manuellen Testfälle auf zuvor definierte Funktionsbausteine und VBA-Routinen in QTP aufzuschlüsseln und auszuführen.

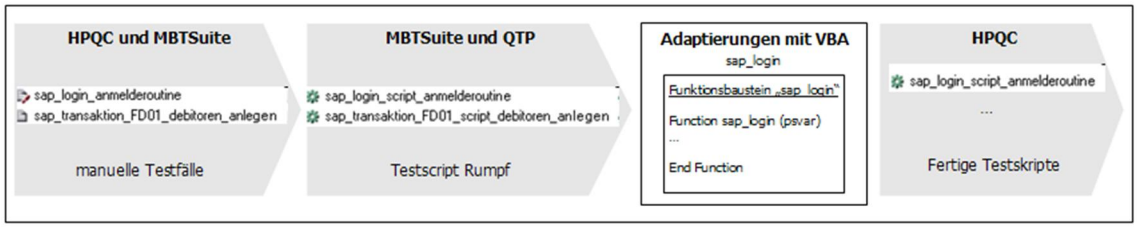


Abbildung 67: Vorgehensweise bei der Testfallermittlung – Adaptierung

Dieser Ansatz schließt die Lücke bisheriger MBT-Methoden [98], [99] und ermöglicht über den „Umweg" der Adaptierung die Generierung fertig automatisierter Testskripte [2], [116]. Die so entstehenden Programmstücke (Adaptierungen) sorgen dafür, dass mit Hilfe von Namenskonventionen immer nur die Funktionen in der Funktionsbibliothek aufgerufen werden, welche zum Namen des aufzurufenden Testskripts passen. Dazu war es nötig, einen Adapter zu programmieren, der die Fähigkeit hat, alle Funktionen im Testautomatisierungs-Repository auszulesen und zu vergleichen. Gleichzeitig muss die Adaptierung eine Referenz vom Testskript auf die hinterlegten Testlogiken in der Funktionsbibliothek mittels eines VBA-Befehls setzen.

4.1.1 Testwerkzeuge als Voraussetzung einer MBT-Methode

Neben den schon vorgestellten zu testenden Softwaresystemen und den Testwerkzeugen bilden die Testdaten und deren Qualität die Basis für die erfolgreiche MBT-Integration in einer Unternehmensorganisation. Für MBT ist ein etabliertes und funktionierendes Testvorgehen im Unternehmen Voraussetzung, da MBT eine Erweiterung des klassischen Tests darstellt [140].

Bei der Betrachtung, welche Testwerkzeuge in Frage kommen, ist zu berücksichtigen, dass die Zielgruppen (welche die Testfälle modellieren) keine technisch versierten Ingenieure sind, sondern Benutzer mit dem Schwerpunkt der Gestaltung von End-to-End-Geschäftsprozessen im Unternehmen. Die richtige Auswahl von Tools spielt eine wichtige Rolle für das Gelingen der MBT-Lösung [30]. Hier gilt es, auf softwareunabhängige[87] Testwerkzeuge zu setzen, da integrierte Geschäftsprozesse zu testen sind, an denen neben ERP-Systemen wie SAP weitere Drittsysteme beteiligt sind.

Sowohl für das Test- als auch für das Requirement-Management bietet der Marktführer im Testmanagement, HP ALM („Hewlett Packard Application Lifecycle Management"), eine breite Palette von Anwendungsmöglichkeiten für das Applikationsmanagement [139]. Es beinhaltet das Testmanagementtool HPQC (Quality Center) für die Organisation und die Verwaltung manueller und automatisierter Tests. Zusätzlich bietet es das Automatisierungstool Quicktest Professional (QTP), welches zum Automatisieren von Tests eingesetzt wird [161]. Das HPQC hat spezielle Adapter, die es erlauben, verschiedene Testwerkzeuge anzubinden, wie bspw. Modellierungswerkzeuge und Testfallgeneratoren. So kann das HPQC für das MBT als Testmanagementtool genutzt werden.

[87] Es gibt Testtools, die sich auf bestimmte Softwaresysteme spezialisiert haben, aber bspw. bei Webapplikationen auf Probleme stoßen. http://greiterweb.de/spw/test_werkzeuge.htm.

4.1.2 Ist-Analyse der eingesetzten Testwerkzeuge in der Praxis

In den letzten Jahren kamen verstärkt so genannte „Application Lifecycle Management (ALM)"-Werkzeuge auf den Markt. Diese haben den Anspruch, die verschiedenen Aktivitäten des Entwicklungsprozesses ineinander verzahnt zu unterstützen. Dieser ganzheitliche Ansatz findet sich sowohl in den klassischen Software-Vorgehensmodellen als auch in der agilen Softwareentwicklung wieder. ALM-Werkzeuge existieren in verschiedenen Ausführungen, zum einen sind da komplette Neuentwicklungen, die eher für den agilen Ansatz sprechen, wie [145], [146].

Für diese Dissertation und das betrachtete Fallbeispiel wird HP ALM (Hewlett Packard) genutzt. Es ist ein weitverbreitetes Testwerkzeug [118] und bietet mit seiner Funktionalität den Testteams einen Testprozess aus einer Hand (mit einem Tool). Zusätzlich bietet das HPQC seit der Version 9.2 umfangreiche Funktionen wie ein Requirement-Management. Als eines der ersten Testwerkzeuge unterstützt es ein integriertes Requirement-Engineering sowie ein Release-Management und die Testausführung bis zur Testauswertung auf einer Plattform. Kurzgefasst unterstützen die HPQC-Module den kompletten Softwareprozess der Entwicklung bis zur Kundenabnahme, einschließlich Testausführung und Fehlerbehebung. Dadurch erhalten Unternehmen neue Möglichkeiten für ein wirkungsvolles „Application Lifecycle Management", von denen sowohl das Qualitätsmanagement, die Entwicklung als auch die Projektleitung stark profitieren. Testinformationen sowohl zur Testausführung als auch im Fehlerhandling werden in den jeweiligen Modulen erfasst.

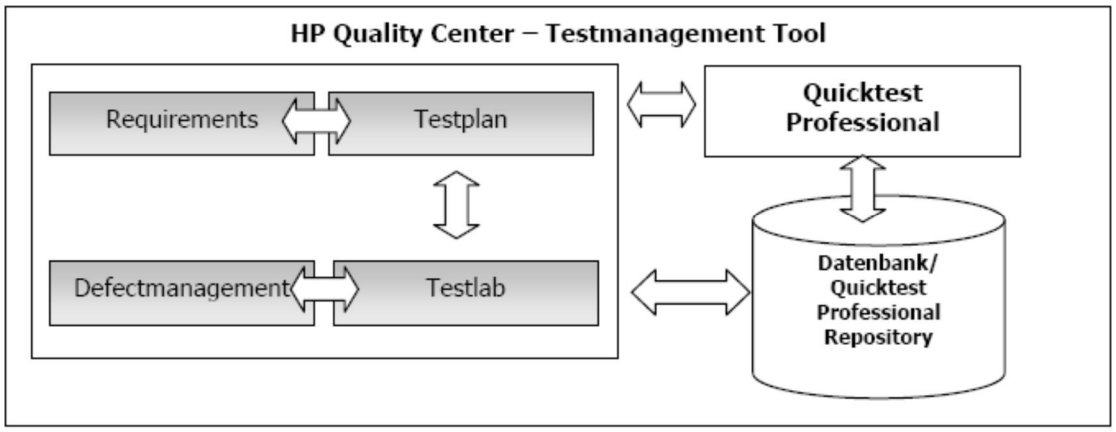


Abbildung 68: Das HPQC als Testmanagementwerkzeug und am Lösungsansatz beteiligte Komponenten

Im Testplan werden Testfälle definiert und gleichzeitig spezifiziert, im Testlab (Testlabor) werden verschiedene, zusammengehörende Testfälle in Testläufen, so genannten „Test Sets", organisiert. Diese werden in Baumstrukturen abgelegt, dabei sollte die Struktur nach Softwaresystemen und Softwareversionen (Releasenummerierung) gegliedert werden. Auch werden gleichzeitig die Testfälle den Anforderungen aus dem Requirementmodul zugeordnet. Unter Defects[88] (Defectmanagement) werden gefundene Abweichungen (Fehler) festgehalten und verwaltet. Durch die Zuordnung zu Testfällen können somit detaillierte Auswertungen zu Fehlerschwerpunkten gemacht oder Trends analysiert werden. Der große Vorteil des HPQCs liegt in der Webfähigkeit, so können weltweit Testteams gemeinsam an Testprojekten arbeiten. Deswegen bietet das HPQC eine professionelle Administration von Benutzern, so werden Zugriffsrechte über ein Administrationsmodul vergeben und verwal-

[88]Abweichung zwischen Soll und Ist.

tet [139]. Zusätzlich können verschiedene Benutzerkonten und Benutzerrollen angelegt werden. Klassische Rollen sind bspw. der Entwickler (Entwicklerrolle, mit der Entwickler Modultests erstellen und ausführen dürfen) oder der Anwender aus dem Fachbereich, der für den Kundenabnahmetest verantwortlich ist. Das HPQC bietet zudem die Möglichkeit, Planungsvorgänge von Tests aktiv zu unterstützen [139]. Eine Gliederung aller geplanten Tests nach verschiedenen Gesichtspunkten kann dafür sorgen, dass der Testprozess leichter zu überschauen und damit auch zu verwalten ist [118]. Es ist nicht nur eine Einteilung nach den zu testenden Produkten, wie bspw. SAP und den dazugehörigen Teststufen, möglich, sondern es können im HPQC weitere auswertungsrelevante Faktoren (mit Hilfe des „Test Subject Tree“) angezeigt werden [139].

Der „Test Subject Tree“ wird gleichzeitig in [67] spezifiziert. In den Ästen dieses „Test Subject Tree“ stehen einzelne konkrete Testfälle, somit können Auswertungen anhand von Kriterien vorgenommen werden, wie z. B. der Aspekt, welche Testfälle sich auf Bildschirmmasken beziehen und welche Testverfahren dabei eingesetzt und durchgeführt werden. Ein weiteres Testwerkzeug, welches aus der HP-Produktfamilie stammt, ist Quicktest Professional (QTP). Es ist ein Testwerkzeug zur automatisierten Durchführung von Tests aller Teststufen. Die Skripte basieren auf der Sprache Visual Basic (VBA). Konzipiert wurde QTP für Windows, Oracle, SAP und für verschiedene webbasierte Anwendungen. Zusätzlich zu den Kernfunktionalitäten von QTP gehören die ((Capture & Replay) Aufnahme und Wiedergabe von Testschritten)-Funktion und die Definition von Verifikationspunkten, welche benötigt werden, um Werte, Variable und das korrekte Arbeiten der zu testenden Anwendung unter Testbedingungen zu überprüfen.

4.1.3 HPQC und QTP als Entwicklungsumgebung

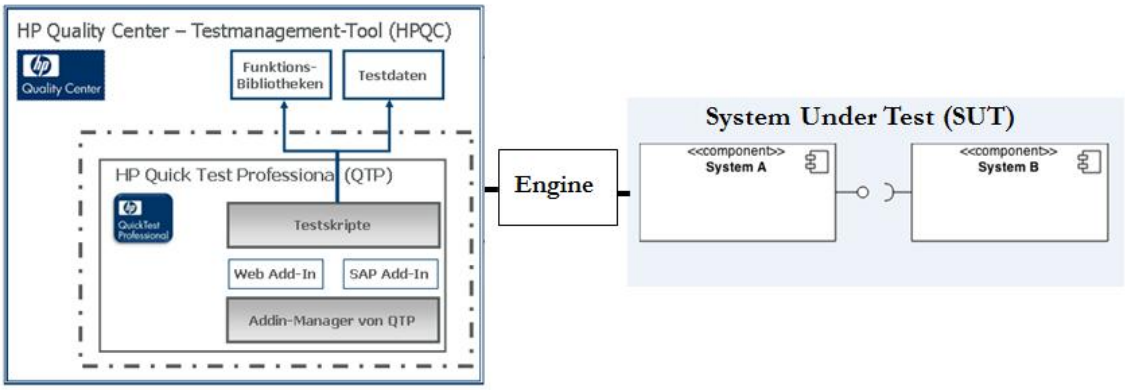


Abbildung 69: HPQC und Ablagestruktur für Testskripte und Testdaten

Das HPQC ist als Testmanagementtool zu verstehen, in das mehrere Testwerkzeuge integriert werden können. QTP ist ein Tool, welches die Automatisierung von Testfällen (auch GUI[89]-Tests unter Windows) unterstützt. Sowohl die internen Repräsentationen von internen Windows-Steuerelementen als auch SAP (mit Hilfe einer selbst zu entwickelnden Bibliothek), Oracle und Java-Anwendungen können identifiziert werden. Zusätzlich bietet QTP das Aufzeichnen von Testschritten, um die aufgezeichneten Programmelemente mittels Parametrisierung, Programmierschleifen und Funktionsbausteinen zu erweitern.

[89]Graphical User Interface.

Neben dem Testfall und dem Testskript spielen die Testdaten und die Funktionsbibliotheken sowie eine festgelegte Ablagestruktur für Testfälle und Testskripte eine zentrale Rolle [118]. Die Funktionsbibliotheken beherbergen die Programmlogik (Testablauf) eines Testfalls, das Testskript selbst beinhaltet eine Reihe von Prozeduraufrufen hinsichtlich der in der Funktionsbibliothek für den Test festgelegten Routinen [139]. Sowohl das HPQC als auch QTP bilden in der MBT-Werkzeugkiste die ersten Basisbausteine. Im nächsten Schritt müssen jetzt sowohl ein Modellierungswerkzeug als auch ein Testfallgenerator ausgewählt werden, welche sich in die hier vorgestellten Testwerkzeuge von HP integrieren lassen.

4.1.4 Requirements im HPQC

Das HPQC wird oft nur als Testmanagement-Werkzeug wahrgenommen [150]. Bisher unbekannt ist, dass das Tool jedoch bereits seit einigen Jahren auch die Möglichkeit bietet, Requirements zu verwalten. Das Requirement-Modul ist im HPQC integriert und ermöglicht die Erweiterung auf Geschäftsprozessmodelle [118], [96]. Es ermöglicht zusätzlich die Integration von Anforderungen und Tests, so dass verteilte Teams auf Basis zentraler Anforderungen zusammenarbeiten und Beziehungen zwischen Tests in Verbindung mit Defects besser auswerten können. Dies erfolgt anhand einer gemeinsam genutzten Repository.

HP ALM ermöglicht es, Abhängigkeiten unter Anforderungen zu verlinken und diese nahtlos mit Releases, Testfällen, Testergebnissen, Quellcode und Defects zu verknüpfen. Mit solchen Verknüpfungen werden auch Benachrichtigungen gesteuert, welche die Benutzer über den aktuellen Stand ihrer Anforderungen unterrichten. So kann u.a. festgelegt werden, dass der Autor eines Testfalls bei einer geänderten Anforderung per E-Mail benachrichtigt wird. Diese Abhängigkeiten können mit verschiedenen Mitteln dargestellt (Tracability Matrix) und analysiert werden (Impact Analysis) [139], [118]. Anforderungen können als „User Stories“ im Requirementmodul erfasst werden. In der Praxis ist es gelebter Standard, direkt die Anforderung als funktionale Spezifikation zu erfassen und daraus je Funktionseinheit einen Testfall zu bilden [118], [157].

Die Organisation der Anforderungen erfolgt in den Backlog-Containern. Ausgehend von den einzelnen abgeleiteten Funktionseinheiten werden dann Tests entwickelt und modelliert. Das HPQC erlaubt es, die Funktionseinheiten mit den generierten Testfällen und Testskripten zu verbinden, und somit kann die Abdeckung der Funktionalität von Anforderungen mittels Testfällen überwacht werden.

4.1.5 Testdaten und die definierte Ablagestruktur

Sowohl für die manuelle als auch für die automatisierte Testausführung kann das HPQC mit all seiner Funktionalität genutzt werden. Für die Erweiterung der manuellen Testfälle zu ausführbaren Testskripten müssen, wie schon erwähnt, Adaptierungen vorgenommen werden. Das entwickelte Adaptierungsprogramm erhält dadurch Zugriff auf den abstrakten Syntaxbaum von QTP. Mit Hilfe von festgelegten Namenskonventionen können dann zu den manuell generierten Testfällen korrespondierende Funktionsbausteine (VBA-Code) angefertigt werden, welche auf Schlüsselwörter reagieren. Die so zu jedem Softwaresystem einzeln codierten Funktionsbausteine müssen danach zu einer Funktionsbibliothek zu-

sammengefasst und im HPQC allen am Test beteiligten Testskripten zur Verfügung gestellt werden.

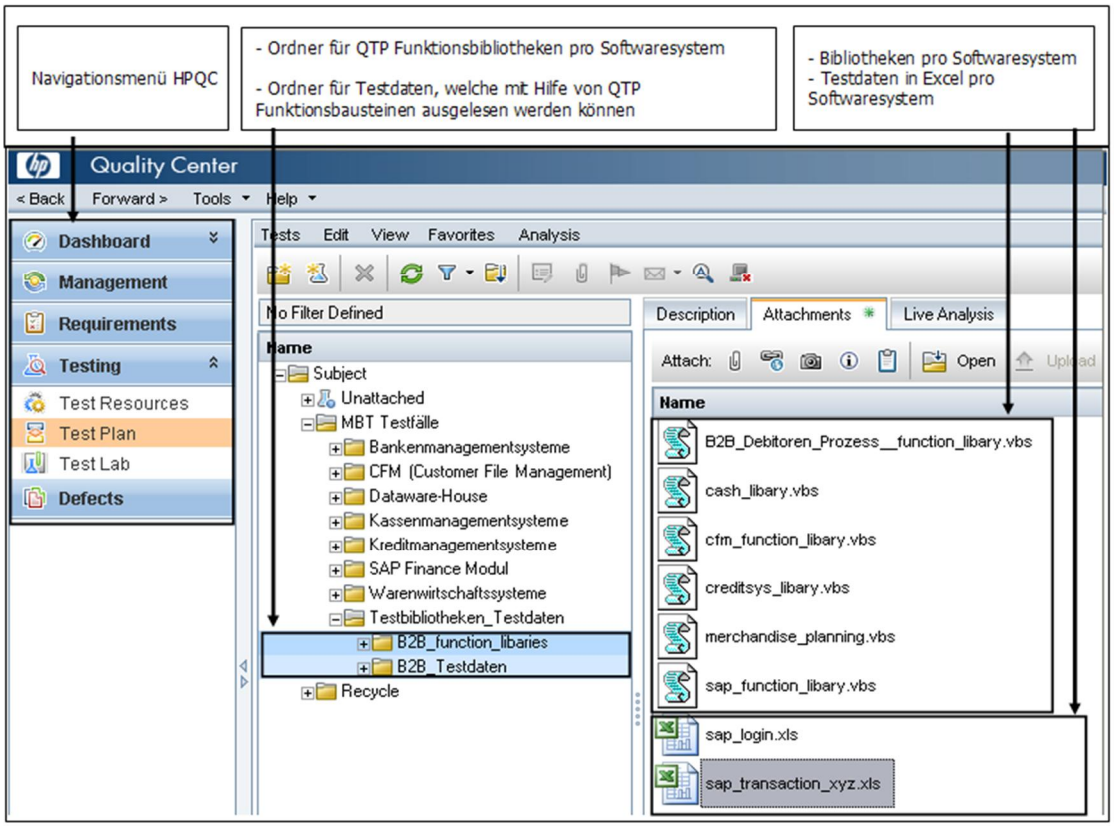


Abbildung 70: Ablagestruktur – Funktionsbibliotheken und Testdaten

Das kann erreicht werden, indem für jedes Softwaresystem ein eigener Testordner angelegt wird, in welchen dann aus der MBTSuite die generierten Testfälle übertragen werden. Zusätzlich werden noch zwei Ordner (B2B_Testdaten, B2B_function_libaries) angelegt, im ersten werden Testdaten (B2B-Testdaten) und im zweiten die programmierten Funktionsbausteine (B2B-function_libaries) abgelegt.

Unabhängig davon, ob ein Testlauf automatisiert oder manuell ausgeführt werden soll, müssen Testergebnisse und Fehlermeldungen im HPQC dokumentiert werden. Damit schafft das Testmanagementwerkzeug Transparenz, weil es von der Testanforderung über das Testmodell bis hin zur Testausführung immer eine Dokumentation der einzelnen Phasen unterstützt [145]. Im Fehlerfall kann so immer von einem Fehler bis zur Testanforderung der einzelnen Funktionseinheit zurücknavigiert werden [118], [149]. Testdaten können im Excelformat oder direkt in einem Testmodell mit Hilfe von UML-Klassendiagrammen hinterlegt werden. Des Weiteren werden die Testdaten mittels Äquivalenzklassenbildung spezifiziert, welches gleichzeitig für die repräsentative Testfallableitung dient. Die Grenzwertanalyse nimmt die Äquivalenzklassen als Basis, um die Ränder zu testen, weil die Erfahrungen aus der Praxis zeigen, dass genau dort die Fehler liegen können [9].

4.1.6 Testfallmodellierung mittels IBA und BPMN als Spezifikationssprache

Die MID GmbH bietet mit IBA ein neues Modellierungstool, das speziell für die Zusammenarbeit zwischen Fachbereich und IT konzipiert ist [121]. Es unterstützt das breite Spektrum der Modellierung von Geschäftsprozessen auf unterschiedlichen Ebenen. Als Notation wird BPMN 2.0 eingesetzt, wobei auch UML-Klassendiagramme und Use Cases unterstützt werden. IBA implementiert die BPMN-Notation im vollen Umfang. Erprobt wurde das Modellierungswerkzeug im Rahmen dieser Dissertation als Testmodellierungswerkzeug. Ein großer Vorteil von IBA ist, dass sich das Werkzeug nicht am OMG-Standard orientiert, sondern dem jeweiligen Anwender im Fachbereich die für seine Rolle notwendigen Elemente bereitstellt. Somit werden die Anwender mit Hilfe der Erweiterung der BPMN-Spezifikation tatsächlich nur mit den für die Testmodellierung notwendigen Elementen konfrontiert. Dadurch werden sie nicht unmittelbar mit der Mächtigkeit von BPMN konfrontiert. Die Einfachheit und die leichte Bedienung von solchen Werkzeugen sind wichtige Faktoren bei der Akzeptanz und der Einbindung der Fachbereiche in das Thema Testen.

Ein Business-Analyst übernimmt die Modellierung, wobei er als Mittler zwischen Fachbereich und IT zu verstehen ist. Er muss sowohl über IT-Wissen als auch über Prozesskenntnisse im jeweiligen Unternehmen verfügen. So muss der Business-Analyst in der Lage sein, IT-Anforderungen aus Unternehmensfachbereichen aufzunehmen und diese zusammen mit einem IT-Experten so abzustimmen, dass ein Testmodell für die gestellte Anforderung angefertigt werden kann. Wichtig ist, dass ein Business-Analyst die Testprozesse aus Sicht des Anwenders modelliert. Er muss das Systemverhalten analog zum Produktivbetrieb simuliert im Testmodell darstellen. Viele Unternehmen haben keinen speziell ausgebildeten Mittler zwischen Fachbereich und IT [121]. Umso wichtiger ist es, bei der Einführung einer MBT-Testmethode diese Mittlerrolle zu schaffen. Eine im Rahmen dieser Dissertation vorgenommene Untersuchung der MBTSuite in Kombination mit IBA brachte hervor, dass Fachbereiche, Business-Analysten und IT-Experten eng zusammenarbeiten müssen, um das gewünschte Systemverhalten tatsächlich auch in einem Testmodell abzubilden [85]. Zusätzlich müssen die Testmodelle mit Äquivalenzklassen erweitert werden. Die Detaillierung von in BPMN modellierten Testdatenelementen kann durch UML-Klassendiagramme vorgenommen werden. Da IBA lediglich ein Modellierungswerkzeug ist und keine Ausführungseinheit anbietet, muss demzufolge ein mittels IBA angefertigtes Testmodell in ein externes Werkzeug exportiert werden, um ausgeführt zu werden.

Die Ablagestruktur ist für die Testautomatisierung eine Komponente, um Testskripte zu identifizieren. So ist es für ein Testautomatisierungswerkzeug wie QTP wichtig zu wissen, in welchen Ordnern und Dateien sich die für den Test erforderlichen Testdaten und Funktionsbausteine befinden.

4.1.7 Die MBTSuite

Die MBTSuite ist ein Testwerkzeug zur automatischen Generierung von Testfällen aus Modellen. Das Werkzeug bildet die Brücke zwischen Modellierungs-, Testmanagement- und Testautomatisierungswerkzeug. Es unterstützt verschiedene Testfallerzeugungsstrate-

gien, da sich Testziele abhängig von der zu testenden Software unterscheiden können. Angefangen bei den im Testmodell vorgegebenen Pfaden, bis hin zu vollständigen Abdeckungsmaßen, wie Kanten- oder komplette Pfadüberdeckung. Verteilte betriebliche Informationssysteme (BIS) zeichnen sich durch komplexe Geschäftsprozesse aus. Um dabei aber eine Explosion von Testfällen zu vermeiden, können in der MBTSuite Konfigurationen vorgenommen werden, welche Einschränkungen (Filtereigenschaften) von Schleifendurchgängen usw. enthalten. So ist sichergestellt, dass nur die Testfälle generiert werden, welche bspw. als geschäftskritisch gelten und für eine Prozessüberdeckung benötigt werden.

Ein Abdeckungskriterium, welches vom Autor dieser Dissertation anvisiert wurde, ist die Prozessabdeckung im Test verteilter betrieblicher Geschäftsprozesse. Die Prozessabdeckung basiert auf den Pfadabdeckungen, jeder Pfad entspricht im Modell einem Testschritt, wobei es nicht entscheidend ist, alle möglichen Pfade im Test zu durchlaufen, sondern nur die für den geschäftskritischen Prozess notwendigen Pfade. Bei einer Prozessüberdeckung werden die geschäftskritischen Pfade im Testmodell generiert. Markiert sind diese mit der Priorität „1“ und „2“. Die Prozessabdeckung ist bisher kein Messkriterium für Testabdeckungsgrade und kann in diesem Falle nur im Ermessen der Modellierer, gemeinsam mit dem Fachbereich, liegen. Bei der Analyse der Regressionstestfälle müssen alle Priorisierungen im Testmodell spezifiziert werden. Somit ist sichergestellt, dass bei der Generierung alle geschäftskritischen Prozesse und somit auch eine Prozessüberdeckung gegeben sind.

Mit Hilfe von IBA ist es dann möglich, Geschäftsprozesse zu modellieren und gleichzeitig alle Pfade, die eine Software durchlaufen kann, darzustellen [121]. Die Einbindung des Modells in die MBTSuite erfolgt über eine offene Schnittstelle, welche die MBTSuite anbietet [85]. Das macht die MBTSuite sehr flexibel hinsichtlich der einzusetzenden Modellierungswerkzeuge.

4.2 Testfallspezifikation als Grundlage für die BPMN-Testmodellierung

Das zentrale Problem des Testens ist die Festlegung der relevanten, zu testenden Eigenschaften. Die zu testenden Eigenschaften in einem Softwaresystem müssen schon während einer Anforderungsspezifikation identifiziert werden, denn sie sind spezifisch für das zu testende Softwaresystem [11]. In den folgenden Unterabschnitten werden schrittweise die Spezifizierungen der einzelnen Testfälle erläutert, welche dann zum IT-gestützten Zahlungsabwicklungsprozess (Debitoren-Prozess) zusammengefügt werden. Wobei hier der Testfallprozess als Prozessablauf beschrieben werden soll. Sowohl Äquivalenzklassenbildung als auch Grenzwertanalysen und Testvarianten können dann analog im Testmodell [85] spezifiziert werden. Mit dem hier beschriebenen Lösungsansatz muss im Testmodell hinterlegt werden, welche Datenformate und Dateninhalte zwischen Softwaresystemen übertragen werden sollen. Als Testdatenbehälter wurde eine Kombination zwischen UML-Klassendiagrammen und Dateien im Excelformat vorgeschlagen.

Die am Fallbeispiel beteiligten Softwaresysteme beschreiben Prozesse, welche wiederum aus mehreren Unterprozessen bestehen können. Damit die Modelle trotzdem übersichtlich

bleiben, müssen für jedes dieser Softwaresysteme sowohl einzelne Testspezifikationen als auch einzelne Testmodelle angefertigt werden.

Der abstrakte Standardablauf im IT-gestützten Zahlungsprozess [142], [33] ist wie folgt:

1) Neukunden anlegen im CFM – das System generiert zum angelegten Kunden eine Kundennummer. Gleichzeitig wird zum angelegten Kunden die Kundenkarte gedruckt und ausgehändigt. Diese ermöglicht es ihm, bei B2B-Unternehmen einzukaufen.
2) Neukunden im Kreditmanagementsystem auf Bonität prüfen und gleichzeitig die zum Neukunden erfassten Stammdaten ins Buchhaltungsprogramm SAP-FI übertragen.
3) Der Neukunde muss gleichzeitig als Debitor angelegt werden. Das SAP-FI-Buchhaltungsprogramm generiert eine Debitorennummer, welche 1:1 einer Kundennummer aus dem CFM-System zugeordnet wird. Dazu erhält es vom CFM den Kundendatensatz.
4) Das Ergebnis aus der Bonitätsabfrage muss vom Kreditmanagementsystem sowohl an die Kassensysteme als auch an das CFM und an die Buchhaltungssysteme übertragen werden.
5) Kassenprozess starten, indem zur Simulation Produkte eingekauft werden; prüfen, ob die Kassensysteme die Daten korrekt anzeigen und verarbeiten.
6) Das Prüfen, ob der Kunde auch im Kassensystem angelegt ist, geschieht parallel zum Scanprozess der Produkte. Prüfen, ob vom Kreditmanagementsystem die korrekten Werte in die Kassensysteme (Euro) übertragen worden sind.
7) Nach Kassenabschluss prüfen, ob die Daten korrekt an die Buchhaltungssysteme, an das CFM und an die Warenwirtschaftssysteme übertragen werden.
8) Prüfen des Bestandes im Warenwirtschaftssystem auf der Basis der von der Kasse gemeldeten Artikel und Produkte.
9) Den Kernprozess „Zahllauf“ des IT-gestützten Zahlungsprozesses starten, prüfen, ob die Lastschriften verarbeitet werden. Dazu werden Bankenprozesse ausgelöst. Analog dazu werden „Mahnläufe“ bei Zahlungsausfall gestartet.
10) Prüfung der Aktualität der Kreditgewährung sowohl im Kassensystem als auch in der Buchhaltung.
11) Wenn Mahnstufe 3 erreicht ist, Kunde sperren, oder wenn der Zahlprozess erfolgreich ist, Kunde automatisch wieder entsperren.

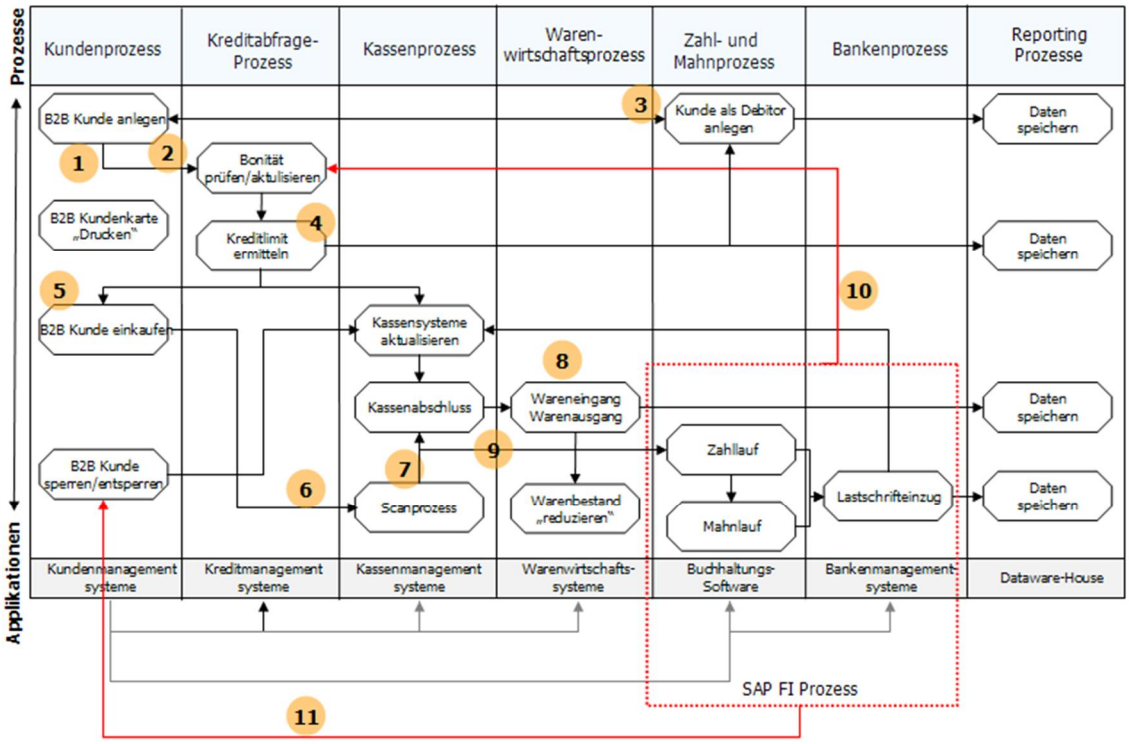


Abbildung: 71 Prozessablauf aller beteiligten Softwaresysteme

Gestartet werden soll mit dem Kundenmanagementsystem, welches in der Praxis auch CFM (Customer File Managementsystem[90]) genannt wird. Für die Demonstration des „Best Practice"-Ansatzes wird das Produkt Siebel benutzt. Als Initialprozess wird im CFM ein Neukunde angelegt.

4.2.1 Testfallbeschreibung Kundenmanagementsystem (CFM)

Bei der Neuanlage eines Kunden müssen Vorbedingungen berücksichtigt werden, welche sich in den Testfällen widerspiegeln. Bei B2B-Unternehmen ist es nur Händlern gestattet, im jeweiligen Unternehmen einzukaufen. Damit dies sichergestellt ist, werden berechtigte Einkäufer mit einer Kundenkarte ausgestattet. Einkäufer sind Kunden, welche wiederum Unternehmer (Hotelketten, Gastronomen usw.) sind.

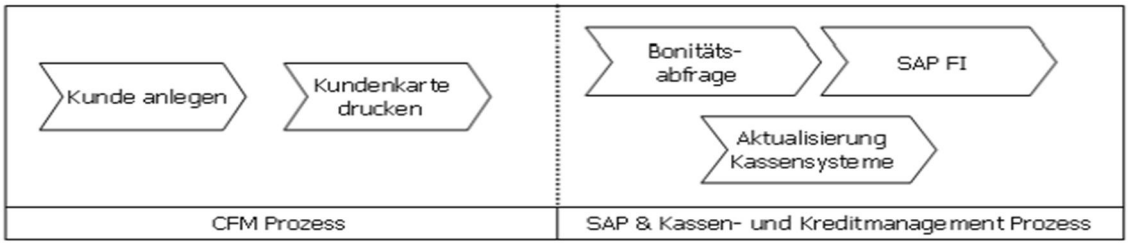


Abbildung 72: Prozessfolge vom CFM zu angebundenen Softwaresystemen (SAP-FI, Kassensystem, Kreditsystem)

Damit dieser Kunde im CFM angelegt werden kann, muss der Prozess „Kunde anlegen" ausgeführt werden. Dazu gibt es im B2B mehrere Möglichkeiten. Für das Fallbeispiel im Rahmen dieser Dissertation ist es ausreichend, einen Kunden auf der Basis von Kundenstammdaten anzulegen.

[90] Siebel-Produktpalette – dieses Unternehmen benennt die Kundenmanagementlösung.

4.2.1.1 Prozess des Kundenanlegens

Damit ein Kunde angelegt werden kann, müssen Benutzer über ausreichend Berechtigungen im CFM-System verfügen. Das setzt voraus, dass ein Benutzer im CFM-System existiert und gleichzeitig mit ausreichend Berechtigungen ausgestattet ist. Zusätzlich müssen die für einen Test notwendigen Testdaten zur Verfügung stehen, denn nur mit Testdaten ist ein Test erst möglich. Nachdem diese beiden Voraussetzungen erfüllt sind, kann mit dem „eigentlichen" Anlegen des Kunden begonnen werden. Dazu müssen mit dem CFM nacheinander Transaktionen aufgerufen und Testdaten eingepflegt werden. Testfälle müssen, um übersichtlich und wiederverwendbar zu bleiben, modular aufgebaut sein, sie können als Vorlage für die automatisch zu generierenden Testfälle aus Modellen dienen. So können schon im Modell Festlegungen gemacht werden, welche es erlauben, die Testfälle beim Generieren nach Bausteinen modular aufbauen zu lassen.

- Baustein 1: Anmelderoutine beim Softwaresystem (beliebiges Softwaresystem) – Anmelderoutinen unterscheiden sich nicht wesentlich voneinander.
- Baustein 2: Prüfen von Berechtigungen im zu testenden Softwaresystem. An dieser Stelle können Testautomatisierungswerkzeuge unterstützend wirken. Programmierte Testskripte können korrespondierend zu einem Benutzer prüfen, ob er Transaktionen und Programme aufrufen darf, welche es ihm erlauben, überhaupt einen Test durchzuführen. Beim Prozess „Kunden anlegen" können bspw. die Transaktionen zum Kundenanlegen getestet werden.
- Baustein 3: Vorbedingungen für das Testen schaffen (Initialisierung von Testdaten). Auch hier kann ein Testautomatisierungswerkzeug dazu dienen, Initialdaten vor jedem Testdurchlauf anzulegen. Beim Prozess „Kunden anlegen" könnte das bspw. das Anlegen von verschiedenen Branchen oder Branchengruppen sein.
- Baustein 4: Testdurchführung (Aufrufen von Transaktionen, Programmen und das Pflegen von Feldern mit spezifizierten Testdaten). Dieser Baustein enthält die softwarespezifische Testlogik und gleichzeitig alle notwendigen Testschritte sowie die jeweils pro Testschritt zu erwartenden Ergebnisse. Die zu pflegenden Parameter sind fett markiert, >> ? << im Testfall steht für einen Platzhalter.
- Baustein 5: Abbauen der Verbindungen, Schließen der zu testenden Software. Dies soll dafür sorgen, dass für den Test blockierte Ressourcen wieder freigegeben werden. So können evtl. zur Testlaufzeit im Baustein 3 und 4 aufgebaute Verbindungen zu anderen Softwaresystemen usw. wieder abgebaut werden.

Die folgende textuelle Testfallspezifikation dient ausschließlich bei der ersten Einführung als eine MBT-Methode, um für die Modellierer die Testfallmodellierung so einfach wie möglich zu gestalten. Steht erst einmal eine konsistente und mit den Fachbereichen abgestimmte Spezifikation, so kann ein Testfallmodell angefertigt und sukzessive erweitert werden.

Ein Beispiel, welches demonstrieren soll, wie die einzelnen Bausteine in Testfällen abgebildet werden:

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 1	„Aufrufen" des CFM Programms	CFM-„Anmeldemaske" erscheint

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 2	Eingeben: Standortkennziffer >> ? << [String] und Marktnummer >> ? << [String]	Wertehilfe lässt sich „aufrufen“ und für den Test erforderliche Werte lassen sich „selektieren“
Schritt 3	Benutzername >> ? << [String] Passwort >> ? << [String] eingeben und auf „anmelden“ klicken	Anmeldung in CFM erfolgt – „Einstiegsbildschirm“ erscheint
Schritt 4	Aufrufen der Siebel-Transaktion „*K00F09*“	„Kunden anlegen“-Maske erscheint
Schritt 5	Kunden-anlegen-Maske „schließen“	„Einstiegsbildschirm“ erscheint
Schritt 6	Prüfen, ob Testdaten vorhanden sind: Marktnummern, Kundengruppen, Rechtsformen, Branchen Prüfen, ob Kunde Anforderungen erfüllt, wie: Gewerbeanmeldung und Steuernummer und ein gültiger Personalausweis.	Wertehilfen erscheinen und das Selektieren von Werten ist möglich
Schritt 7	Daten „sichern“ und CFM „beenden“	CFM-„Einstiegsbildschirm“ schließt

Mit Schritt 3 und 4 werden grob die beiden Bausteine 2 und 3 angewandt. Das eigentliche Testen beginnt mit Baustein 4. Schritt 2 wird mit Baustein 1 abgedeckt. Die Testaktivitäten werden dann konkret in Baustein 4 abgewickelt, wobei dieses obige Beispiel nur darstellen soll, wie der Testfall generell aufgebaut wird. Jetzt beginnt die Testfallspezifikation mit dem Prozess „Kunden anlegen“.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 1	Baustein 1 ausführen	CFM „Anmeldemaske“ erscheint
Schritt 2	Aufrufen der Siebel-Transaktion „K00F09“	„Kunden anlegen“-Maske erscheint
Schritt 3	Nachname: >> ? << [String] Vorname: >> ? << [String] Geburtsdatum: >> ? << [Date] Adresse: >> ? << [String] Hausnummer: >> ? << [String] Wohnort: >> ? << [String] Marktnummer: >> ? << [Integer]	

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
	Personalausweisnummer: >> ? <<[String] Gewerbescheinnummer: >> ? <<[String] Steuernummer: >> ? << [String] Branche: (Auswahlliste F9) >> ? << [String] Kundengruppe: (Auswahlliste F9) >> ? << [String] Rechtsform: >> ? << [String] Rechtsform Gründungsdatum: >> ? << [Date] Bankinformation: >> ? << [String] Kontonummer: >> ? << [Integer] BLZ: >> ? << [Integer] Sperrvermerk >> ? << [String]	
Schritt 4	Datensatz „speichern“	Automatische Kundennummer wird generiert, Datensatz wird gespeichert.
Schritt 5	„Kundenkarte erstellen“ klicken	Formular für elektronische Kundenkarten erscheint

Nachdem der Datensatz angelegt worden ist, folgt im nächsten Schritt (5) die Erstellung der Kundenkarte mit der zeitgleich generierten Kundennummer. Nachdem ein Kunde angelegt worden ist, folgt der Prozess „Kundenkarte drucken“.

4.2.1.2 Prozess „Kundenkarte drucken“ mittels CFM

Die Kundenkarte, die zum Einkaufen berechtigt, wird nach dem erfolgreichen Abarbeiten dieses Prozesses auf einem Kartendrucker gedruckt und dem Kunden überreicht.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 6	Elektronisches Kundenkartenformular „validieren“ und ein Lichtbild des Kunden hinzufügen, Validierung Transaktion „P19“ aufrufen	Kundenkarte gespeichert und freigegeben zum „Druck“
Schritt 7	Aufrufen der Siebel-Transaktion „P90“	Kundenkarte wird ausgedruckt
Schritt 8	Vergleich der Daten auf der Kundenkarte mit den Daten im elektronischen Formular	Kundenkarten-erstellungsmaske „schließen“

Trotz der EU-Datenschutzreform, die das Testdatenmanagement und eine Anonymisierung von Realdaten zu Testzwecken erforderlich macht, nutzen Unternehmen immer noch Realdaten (Produktivdaten) zum Testen ihrer Softwaresysteme.[91] Die Begründung liegt darin, dass diese Daten den tatsächlichen Tagesbetrieb von Unternehmen abbilden. Damit der Gefahr des Datenmissbrauchs entgegengewirkt werden kann, setzen Unternehmen Datenschutzbeauftragte ein, welche insbesondere das Testdatenmanagement unter die Lupe nehmen. Zusätzlich besteht aber neben der Gefahr des Datenmissbrauchs durch Dritte die Gefahr der Unvollständigkeit der Testdaten. So werden die für das Testen notwendigen Äquivalenzklassenbildungen und Grenzwertanalysen nicht berücksichtigt.

Das Fragezeichen zwischen den Klammern signalisiert einen Platzhalter für einen konkreten Testdatenwert. So können zu einem Testfall mehrere Testvarianten angelegt werden. Die Trennung von Testfällen und Testdaten bringt den Vorteil, dass Tests mit mehreren verschiedenen Kombinationen durchgeführt werden können. Der Aufbau (Bausteine 1–5) und der Rumpf eines Testfalls bleiben schematisch gleich. Erst die Testdaten beeinflussen den Testprozess. Auch für die Testautomation bietet die Ausgliederung der Testdaten aus den Testfällen Vorteile [77].

Testdaten können initial auch in Form einer Excel-Tabelle erfasst werden und direkt an ein Testmodell angebunden werden. Das Microsoft-Produkt Excel bietet einige Vorteile, da es sich leicht in Testautomationswerkzeuge integrieren lässt und gleichzeitig auch Fachbereiche im Unternehmen täglich damit arbeiten. Fachbereiche neigen eher dazu, Testdaten in Excel zu erfassen, anstelle von Oracle oder Microsoft Access. Auch diese Testdaten müssen genau analysiert und mit Äquivalenzklassen und Grenzwertanalysen angereichert werden, um so die bestmögliche Testabdeckung zu erreichen.

4.2.1.3 Prozess „Bonitätsprüfung“ mittels starten

Ist die Kundenkarte für den Kunden ausgedruckt, so geht es zum nächsten Prozessschritt, der Bonitätsprüfung des angelegten Kunden. Die Bonitätsprüfung wird mittels CFM gestartet, welches wiederum mehrere Unterprozesse in angebundenen Systemen auslöst.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 9	Bausteine 1–4 ausführen	Kunde mit einer Kundennummer angelegt und validiert, Kundenkarte ausgedruckt
Schritt 10	Prozess „Bonitätsprüfung“ starten	Einstiegsbildschirm erscheint
Schritt 11	Kundenstammsatz per „Freigabe“ an das Kreditmanagementsystem übertragen	Prüfung im Kreditmanagement-system, ob der Datensatz auch angekommen ist

[91] Erfahrungswerte aus der Praxis.

Aus dem CFM-System werden Daten (Kundennummer, Name, Vorname, Anschrift, Geburtsdatum) über die Schnittstelle zum Kreditmanagementsystem übertragen. Diese Daten benötigt das Kreditmanagementsystem, um für einen Kunden das passende Kreditlimit zu berechnen.

4.2.2 Prozess: Testfallbeschreibung aus dem Kreditmanagementsystem

Mit der Übertragung der Daten vom CFM ins Kreditmanagementsystem werden die ersten Schnittstellentests schon implizit durchgeführt. So muss nun überprüft werden, ob die übertragenen Kundenstammdaten mit der richtigen Datenformatierung im Kreditmanagementsystem angekommen sind. Damit der neu angelegte Kunde einkaufen darf, muss zunächst einmal eine Kreditwürdigkeitsprüfung erfolgen. Dazu nutzt das Kreditmanagementsystem die zuvor vom CFM erfassten Daten, welche ihm über eine Schnittstelle übergeben werden. Das Datenaustauschformat zwischen Kreditmanagementsystem und CFM basiert auf XML.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 12	Aufrufen des Kreditmanagementsystems	Kreditmanagement-system-Anmeldemaske „erscheint“
Schritt 13	Anmelden mit Benutzername >> ? << [String] Passwort >> ? << [String]	Kreditmanagement-system-Einstiegsbildschirm „erscheint“
Schritt 14	Im Feld Kundennummer >> ? << [String] die jeweilige Kundennummer des Kunden eingeben, Daten „laden“ klicken	Geladen werden die Stammdaten: Kundennummer [String] Nachname [String] Vorname [String] Marktnummer [Integer] Geburtsdatum [Date] Geburtsort [String] Anschrift [String]
Schritt 15	Kundenabfrage starten, indem der Button „externe Auskunfteien“ ausgewählt wird, und je nach Wunsch die Auskunfteien ausgewählt werden, welche in die Abfrage involviert werden sollen	Fenster mit allen verfügbaren Wirtschaftsauskunftei-Partnern, die im

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
		Kreditmanagement-system verfügbar sind, erscheint
Schritt 16	Nach der Selektion der gewünschten Wirtschaftsauskunfteien durch das Setzen eines Hakens in den „Checkboxen“ kann die Abfrage gestartet werden	Verbindung zu den Wirtschaftsaus-kunfteien wird hergestellt
Schritt 17	Abfrage in Arbeit	
Schritt 18	Ist die Abfrage erfolgreich prozessiert, werden Scorewerte ermittelt, welche durch das Kreditmanagementsystem als Basis für das Kreditlimit genutzt werden	Sanduhr erscheint und Berechnungen werden auf dem Kreditsystem durchgeführt
Schritt 19	Die ermittelten Scorewerte werden im Kreditmanagementsystem markiert und der Button Kreditlimit „berechnen“ im Kreditmanagementsystem wird geklickt	Kundennummer [String] mit all den Stammdaten erscheint zusammen mit dem Betrag, welcher als Kreditlimit [String] diesem Kunden gewährt wird
Schritt 20	Das ermittelte Kreditlimit wird durch das Drücken des Buttons „übernehmen“ an die im Menü verfügbaren angebundenen Systeme übertragen (CFM, Kasse, SAP-FI)	Informationen zum Kreditlimit werden sowohl zu Kassensystemen als auch direkt zurück zum CFM-System und zum SAP-FI-System übertragen

4.2.3 Prozess: Testfallbeschreibung aus dem Kreditmanagementsystem ins SAP-FI

Das ermittelte Kreditlimit wird zusammen mit der korrespondierenden Kundennummer in die Kassensysteme, zu SAP-FI und in das CFM übertragen. In SAP-FI wird dem Kunden zur Kundennummer eine zusätzliche buchhalterische Debitorennummer zugeordnet.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 21	Kundenstammsatz an das angebundene SAP-FI-System übertragen – dazu wird die SAP-	Kundenstammdaten werden dort als

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
	FI-Schnittstelle (Debitorenstammdaten) benötigt	Debitoren angelegt – dazu wird eine Debitorennummer >> ? << [Integer] angelegt
Schritt 22	SAP-FI-Applikationsserver aufrufen	SAP-FI-Anmeldemaske erscheint
Schritt 23	Eingeben der Daten: SAP System >> ? << [String] SAP Client >> ? << [String] Benutzername >> ? << [String] Ländersprache >> ? << [String] Passwort >> ? << [String] Klick auf „Enter“	SAP-FI-Einstiegsbildschirm erscheint
Schritt 24	Aufrufen der Transaktion „FD03“	Debitoren anzeigen, Einstiegsbildschirm erscheint
Schritt 25	Debitorennummer >> ? << [Integer] eingeben und auf „ausführen“ klicken	Allgemeine Daten, Einstiegsbild erscheint, der passende Debitor wird aufgerufen
Schritt 26	Prüfen der Kundennummer vom CFM und der korrespondierenden Debitorennummer im SAP (Passen die im CFM erfassten Daten mit den in SAP übertragenen Daten zusammen?)	Transaktion „FD03“ wird ausgeführt und SAP-Dynpro[92] „Debitoren anzeigen“ erscheint, Abgleich der Daten erfolgreich, Dynpro schließen
Schritt 27	SAP-FI-Modul Kreditmanagement aufrufen durch die Transaktion „F150“	Kreditverfügung zum Debitoren erscheint
Schritt 28	Vergleichen des ermittelten Kreditlimits aus dem Kreditmanagementsystem mit dem Wert im SAP-Kreditmanagementsystem-Modul	Der übertragene Wert aus dem Kreditmanagement-system muss mit dem im SAP-Kreditsystemmodul gespeicherten Wert

[92] Dynamisches Programm.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
		übereinstimmen.

4.2.4 Testfallbeschreibung CFM-Schnittstelle zu Kreditmanagementsystemen

Das Forderungsmanagement in Unternehmen soll gewährleisten, dass erbrachte Lieferungen und Leistungen entsprechend vertraglicher Vereinbarungen bezahlt werden. Dabei unterscheiden Unternehmen zwischen externen und internen Kreditmanagementsystemen. Ein internes Kreditmanagementsystem ist allerdings auch von externen Einflüssen abhängig. Unternehmen nutzen Kreditmanagementsysteme, welche auf sehr komplizierten mathematischen[93] Verfahren basieren. Angebunden ist das Kreditmanagementsystem an externe Wirtschaftsauskunfteien (SCHUFA, CREFO, CEG usw.). Wichtig für das hier zu betrachtende Fallbeispiel ist das an dieser Stelle vom Kreditmanagementsystem zu ermittelnde Kreditlimit (in Euro) [33]. Das Kreditlimit umfasst alle bisherigen Geschäftsbeziehungen des Kunden und seinen bisher erzielten Umsatz für das Unternehmen (bei Neukunden kann es ja noch kein Umsatz geben). Kunden können, je mehr sie einkaufen und wenn sie pünktlich ihre Rechnungen begleichen, ständig ihre Bonität verbessern, so dass das Kreditlimit auch stetig nach oben korrigiert werden kann [33]. Exemplarisch ist hier dargestellt, welche Testdaten über die CFM-Schnittstelle an das Kreditmanagementsystem übertragen werden:

- ✓ Kundennummer [Integer]
- ✓ Vorname[Integer]
- ✓ Nachname [Integer]
- ✓ Geburtsdatum [Date]
- ✓ Geburtsort [String]
- ✓ Adresse [String]
- ✓ Hausnummer [String]
- ✓ Wohnort [String]

```
<Kreditabfrage>
<Kunde>
    <Kundennummer> ? </Kundennummer>
    <Nachname> ? </Nachname>
    <Vorname> ? </Vorname>
    <Geburtsdatum> ? </Geburtsdatum>
    <Geburtsort> ? </Geburtsort>
    <Adresse> ? </Adresse>
```

[93] Auf die im Rahmen dieser Dissertation nicht eingegangen werden soll.

<Hausnummer> ? </Hausnummer>

<Wohnort> ? </Wohnort>

</Kunde>

</Kreditabfrage>

Das Kreditmanagementsystem liest die vom CFM im XML-Format versendeten Daten automatisch ein und verarbeitet diese mit Hilfe von Programmen, welche auf Regeln der Wissenslogik basieren. Kreditmanagementsysteme zeichnen sich zusätzlich dadurch aus, dass Benutzer gesonderte Berechtigungen benötigen. Ein besonderer Grund ist die Sensibilität der Daten, welche sich auf Kreditmanagementsystemen befinden. So dürfen nur ausgewählte Personenkreise (i.d.R. Kundenmanager) Zugriffe auf Kreditmanagementsysteme haben. Unterstützt werden diese Abfrageprozesse durch IT-Prozesse, die sicherstellen sollen, dass ein Vieraugen-Prinzip eingehalten wird.

Im Testfall-Schritt 15 wird dafür gesorgt, dass zu in Schritt 14 geladenen Kundendaten eine Bonitätsabfrage erfolgt. Dazu muss aus dem Kreditmanagementsystem eine Verbindung zu externen Wirtschaftsauskunfteien aufgebaut werden.

In Schritt 16 und 17 werden die Wirtschaftsauskunfteien ausgewählt, zu denen eine Verbindung aufgebaut werden soll, dazu werden einfach „Checkboxen“ aktiviert. Nachdem die erfolgreichen Berechnungen dazu geführt haben, dass Scorewerte für einen Kunden ermittelt worden sind, kann das Kreditmanagementsystem jetzt die eigentliche Kreditlimitberechnung starten. Diese Transaktion dauert i.d.R. wenige Minuten. Als Ergebnis wird auf Basis der Kundennummer die Höhe des zu gewährenden Kreditlimits in Euro ausgegeben. Gleichzeitig werden sowohl die angebundenen Kassenserver, SAP-FI und das CFM bezüglich der Daten und Informationen aktualisiert. Folgende Parameter werden vom Kreditmanagementsystem an die Kassensysteme, an SAP-FI und an das CFM übertragen:

Kundennummer [Integer]

Kreditlimit [String]

<Kreditabfrage>

<Kunde>

<Kundennummer> ? </Kundennummer>

<Kreditlimit> ? </Kreditlimit>

</Kunde>

</Kreditabfrage>

Zumindest beim Kassenserver müssen diese Daten mehrmals am Tag aktualisiert werden [33], [142], um sicherzustellen, dass Bonitätsdaten eines Kunden immer tagesaktuell sind. Die Kassen wiederum besitzen Schnittstellen zu weiteren Softwaresystemen. Im SAP-FI-System werden die übertragenen Kundennummern der jeweiligen SAP-Debitorennummer zugeordnet.

4.2.5 Testfallbeschreibung Kassensystem und angebundene Schnittstelle, Kreditmanagement und Warenwirtschaftssystem

Kassensysteme sind an verschiedene Softwaresysteme angebunden, daher ist es nötig, für jede am Geschäftsprozess beteiligte Schnittstelle einen Testfall zu spezifizieren. Im obigen Testfall-Schritt 20 wurde beschrieben, wie Daten aus dem Kreditmanagementsystem in die angebundenen Kassensysteme und ins CFM übertragen werden. Damit Kassensysteme zur Laufzeit (sprich wenn ein Kunde bezahlen möchte) keine langwierigen Verbindungen zum Kreditmanagementsystem aufbauen müssen, um bspw. Kreditlimits abzufragen, existieren in vielen Kassenlösungen (IBM, Wincor Nixdorf) Zwischenlösungen wie das Abspeichern von Kreditdaten zu einer Kundennummer im Kassensystem. Ein zusätzliches Feld (Kreditverfügung) im Kassensystem wird dann mehrmals täglich vom Kreditmanagementsystem mit Daten aktualisiert. Die Kassensysteme greifen zur Laufzeit dann direkt lokal auf diese Daten zu.

4.2.5.1 Testfallbeschreibung Kassensystem zu Kreditmanagementsystem

Das zentrale Kreditmanagementsystem versorgt sowohl die verteilten Kassensysteme als auch das CFM mit aktuellen Kreditinformationen eines Kunden. Dieser Prozess ist wichtig, da er sicherstellen muss, dass Kreditlimits sowohl nach unten als auch nach oben nicht überschritten werden. Damit jetzt überprüft werden kann, ob tatsächlich die korrekten Kreditlimits in den Kassensystemen angezeigt werden, muss ein Testfall spezifiziert werden, welcher dies aus dem Kassensystem heraus prüft.

Dazu müssen zu Testzwecken von einem Kundenmanager an der Kasse Waren und Produkte eines Kunden in die Kassensysteme übertragen werden. Voraussetzung ist eine Kundenkarte mit einer Kundennummer. Sobald ein Kundenmanager die Kundennummer in die Kassensysteme eingegeben hat, werden sowohl die Kundennummer als auch die Einkaufshistorie des jeweiligen Kunden angezeigt. Ein Kundenmanager kann auf alle Kundeninformationen zugreifen und gleichzeitig auch selbst Entscheidungen treffen. Gleichzeitig kann er zu „alten“ Rechnungen und Aufträgen navigieren und auf Nachfrage vom Kunden diese bereitstellen.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 29	Kassenmanagementsoftware starten und Kassensystem am Kassenclient aufrufen	Anmeldemaske am Kassenserver erscheint
Schritt 30	Anmeldung am Kassenserver über Kassenclient mit Benutzername >> ? << [String] Marktnummer >> ? << [Integer] Sprache >> ? << [String] Passwort >> ? << [String]	Eingaben am Display vollständig, nach dem erfolgreichen Login: Einstiegsbildschirm Kassensystem

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
	auf „Login“ klicken	
Schritt 31	Kundennummer eingeben: Kundennummer >> ? << [String] und auf Button „Kreditverfügung“ klicken	Kunde mit seiner Kundennummer erscheint auf dem Kassendisplay inklusive seiner Einkaufshistorie (falls vorhanden) und dem vom Kreditmanagement-system berechneten Kreditlimit, in welche nur der Kundenmanager Einsicht hat
Schritt 32	Prüfen, ob der in Testschritt 20 vom Kreditmanagementsystem zu diesem Kunden ermittelte Wert entsprechend auch am Kassensystem angezeigt wird	Errechnetes Kreditlimit im Kreditmanagement-system stimmt mit dem im Kassensystem hinterlegten Wert überein.

Ausgehend vom Kassensystem werden weitere Prozesse ausgelöst, z. B. das Warenwirtschaftssystem, das CFM und die Finanzbuchhaltung. Daher ist es notwendig, aus den Kassensystemen heraus mehrere Testfälle zu spezifizieren, welche diese ausgehenden Schnittstellen testen. Das soeben zwischen Schritt 29–32 geprüfte Kreditlimit eines Kunden kann jetzt durch den Kunden in Anspruch genommen werden. Dazu wird nun ein Testfall spezifiziert, der die Erfassung der Produkte durch die Kasse und das Warenwirtschaftssystem abbildet.

4.2.5.2 Testfallbeschreibung Kassensystem – Warenwirtschaftssystem

Der Scanvorgang von Produkten und Waren soll hier nicht Gegenstand der Betrachtung sein, da für diese Tests spezielle Hardware benötigt wird. Dennoch hat die Erfassung der Produkte und Waren an der Kasse einen unmittelbaren Einfluss auf die angebundenen Warenwirtschaftssysteme.

Bei jedem Scanvorgang an der Kasse müssen die eingescannten Produkte vom Warenbestand abgezogen werden. Gleichzeitig überwachen automatische Bestellsysteme den Bestand vieler Artikel. Sobald eine Mindestmenge zu einem Artikel erreicht ist, erfolgt ein automatisierter Prozess, welcher Folgebestellungen zu diesen Artikeln auslöst. Dieser Pro-

zess ist für das Fallbeispiel dieser Dissertation nicht relevant, aber dafür der Prozess vom Kassensystem zum Warenwirtschaftssystem:

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 33	Kassenmanagementsoftware starten und Kassensystem am Kassenclient aufrufen	Anmeldemaske am Kassenserver erscheint
Schritt 34	Anmeldung am Kassenserver über Kassenclient mit Benutzername >> ? << [String] Marktnummer >> ? << [String] Sprache >> ? << [String] Passwort >> ? << [String] dann auf Login „klicken“	Eingaben am Display vollständig und nach dem erfolgreichen Login, Einstiegsbildschirm Kassensystem
Schritt 35	Kundennummer entweder durch ein Kartenlesegerät einlesen oder die Kundennummer über das Kassendisplay im Feld Kundennummer >> ? << [String] eingeben	Kunde mit seiner Kundennummer erscheint auf dem Kassendisplay inklusive seiner Einkaufshistorie (falls vorhanden) und dem vom Kreditmanagement-system berechneten Kreditlimit
Schritt 36	Starte mit dem Erfassen der Produkte Artikel: 100 KG[94] Bratfett Artikelnummer >> ? << [String]	Erfasste Artikel werden auf dem Kassendisplay angezeigt, dazu Artikelnummer >> ? << Menge >> ? << Artikelbezeichnung >> ? <<, Bruttopreis >> ? <<, Nettopreis >> ? <<, enthaltene Mehrwertsteuer >> ? <<, gewährte Rabatte >> ? <<

[94] Kilogramm.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 37	Erfasse mit dem Lasergerät Artikel: Thunfisch Artikelnummer >> ? << [String] Menge >> ? << [String]	Erfasste Artikel werden auf dem Kassendisplay angezeigt, dazu Artikelnummer >> ? <<, Menge >> ? <<, Artikelbezeich-nung >> ? <<, Bruttopreis >> ? <<, Nettopreis >> ? <<, Mehrwertsteuer >> ? <<, und gewährte Rabatte >> ? <<
Schritt 38	Prüfe die im Kassendisplay angezeigten Informationen zu den erfassten Artikeln. Wird die Mehrwertsteuer korrekt berechnet? (7%, 19%) Werden Rabatte prozentual und korrekt angezeigt, bspw. bei Großkunden? Werden die Beträge korrekt berechnet?	
Schritt 39	„Abschluss“ und „Zahlen“ an der Kasse im Display auswählen	Zahlungsvarianten werden angezeigt (Lastschrift, Rechnung, bar, Kreditkarte, EC-Karte)
Schritt 40	„Lastschrift“ auswählen	Rechnung wird gedruckt

Schritt 40 enthält gleichzeitig noch einen versteckten Testfall, denn sollte bspw. ein Kunde keine Kreditgewährung (Bonität) besitzen, so muss das Kassensystem nach Schritt 39 und nach der Auswahl „Zahlen auf Lastschrift“ eine Meldung ausgeben: „*keine Kreditverfügung, nur Barzahlungen möglich*“. So kann an dieser Stelle eine weitere Spezifizierung des Testfalls erfolgen. So kann im Kreditmanagementsystem das Kreditlimit von einem Kunden auf „0 Euro“ gesetzt werden. Zeitgleich wird das neu errechnete Kreditlimit dann an die Kassen-

systeme übertragen. Jetzt sollte bei der Auswahl der Zahlungsart „Lastschrift“ die Meldung „*keine Kreditverfügung, nur Barzahlungen möglich*“ erfolgen. Diese verschiedenen Testvarianten sind Bestandteil der späteren Testfallmodellierung und werden dort berücksichtigt.

Insgesamt sind im obigen Testbeispiel zwei Artikel (100 KG Bratfett mit Artikelnummer >> ? <<, 15 KG Thunfisch mit Artikelnummer >> ? <<) zu einem Kunden erfasst worden. Das Warenwirtschaftssystem erhält vom Kassensystem nach Kassenabschluss eine Information (in Form einer XML-Datei), welche Artikel reduziert werden müssen. Diese Datei wird vom Warenwirtschaftssystem ausgelesen und dementsprechend verarbeitet.

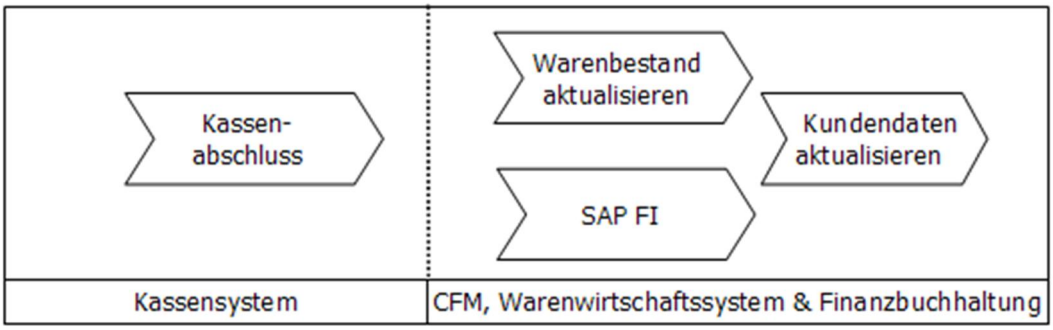


Abbildung 73: Kassensystem – empfangende Systeme

Gleichzeitig werden die Kassenumsätze sowohl nach Kassenabschluss (als XML-Datei) zur weiteren Verarbeitung in die Finanzbuchhaltung als auch in das Kundenmanagementsystem übertragen.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 41	Warenwirtschaftssystem über Client aufrufen und mit Benutzername >> ? << [String] Marktnummer >> ? << [String] Passwort >> ? << [String] anmelden	Warenwirtschafts-system Einstiegsbild-schirm – Explorer erscheint
Schritt 42	Artikelnummer >> ? << [String] eingeben für Bratfett	Artikel Bratfett erscheint in der Anzeige als Listenelement
Schritt 43	Prüfe, ob der Bestand an Bratfett sich nach Einspielung der Kassenabschlussdatei nach unten korrigiert hat	Prüfung erfolgreich, Bestand um 100 KG reduziert
Schritt 44	Artikelnummer >> ? << [String] eingeben für Thunfisch	Artikel Thunfisch erscheint in der Anzeige als Listenelement
Schritt 45	Prüfe, ob der Bestand an Thunfisch sich nach Einspielung der Kassenabschlussdatei nach unten korrigiert hat	Prüfung erfolgreich, Bestand um 15 KG reduziert

Die Daten vom Kassenabschluss werden auch an das CFM gesendet, dort tragen sie dazu bei, Statistiken zum Kunden erstellen zu können. So wird neben der Kreditverfügung auch der generierte Kundenumsatz ins CFM übertragen.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 46	„Aufrufen“ des CFM-Programms	CFM-„Anmeldemaske“ erscheint
Schritt 47	Standortkennziffer >> ? << [String] Marktnummer >> ? << [String] „eingeben“	Wertehilfe lässt sich „aufrufen“ und Werte „selektieren“
Schritt 48	Benutzername >> ? << [String] Passwort >> ? << [String] eingeben und auf „anmelden“ klicken	Anmeldung in CFM erfolgt – „Einstiegsbild-schirm“ erscheint
Schritt 49	Aufrufen der Siebel-Transaktion „K00F37“	„Kundendaten anzeigen“-Maske erscheint
Schritt 50	Kundennummer >> ? << [String] eingeben Geprüft werden jetzt die Umsatzdaten und die korrekte Übertragung der Kreditverfügung. Diese Informationen dienen nur zu statistischen Auswertungen.	Nachname >> ? << [String] Vorname >> ? << [String] Geburtsdatum >> ? << [Date] Adresse >> ? << [String] Hausnummer >> ? << [String] Wohnort >> ? << [String] Marktnummer >> ? << [String] Personalausweis-nummer >> ? << [String] Gewerbeschein-nummer >> ? << [String] Steuernummer >> ? << [String] Branche (Auswahlliste F9) >> ? << [String]

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
		Kundengruppe (Auswahlliste F9)>> ? << [String] Rechtsform >> ? << [String] Gründungsdatum >> ? << [Date] Bankverbindung >> ? << [String] Kontonummer >> ? << [String] BLZ >> ? << [String] Umsatz Einkauf >> ? << [String] Kreditlimit >> ? << [String] Sperrvermerk >> ? << [String] Rechnungsarchiv >> ? << [String] (Auswahlliste F9)
Schritt 51	Daten „sichern“ und CFM „beenden“	CFM schließt

Gleichzeitig wird zwischen den Testschritten 46 und 51 ein Integrationstest zwischen den Schnittstellen Kassensysteme, Warenwirtschaftssysteme und CFM durchgeführt. Das CFM überträgt zusätzlich Daten ins DWH, die hier nicht näher spezifiziert werden sollen. Das DWH bietet Unternehmen generell zahlreiche Analyse- und Auswertungsmöglichkeiten. So können mit dem DWH komplexe Analysen der Debitorengruppen, der Rechnungen und des Kaufverhaltens der Kunden durchgeführt werden. Auf Debitorenebene können anhand der Rechnungsdaten weitere Details pro Artikel erfasst werden. Die vom CFM an das DWH übertragene Datei enthält Daten über offene und ausgeglichene Posten sowie Informationen über angemahnte Posten und Rücklastschriften.

Die Kassensysteme haben neben den Schnittstellen zum CFM, zu den Warenwirtschaftssystemen und zum DWH noch eine weitere zentrale Schnittstelle, nämlich die zur Finanzbuchhaltung SAP-FI.

4.3 Testfallspezifikation ERP-System: SAP-FICO als Grundlage der Testmodellierung

In diesem Abschnitt werden die Schwerpunkte des Testens von IT-gestützten Zahlungsabwicklungsprozessen erläutert. Das zentrale System ist hierbei das SAP-Modul-FI [33]. Dieses empfängt Daten aus Kassensystemen, Bankensystemen und indirekt auch vom CFM und vom Kreditmanagementsystem. Die IT-gestützte Zahlungsabwicklung wird im SAP-FI-Modul vorgenommen, alle anderen am Prozess beteiligten Softwaresysteme leisten ihren Teil dazu, diesen Geschäftsprozess erfolgreich auszuführen. Damit die Testfallbeschreibung zum SAP-FI-Modul sowie zu den angebundenen Systemen nachvollziehbar wird, müssen aus Kapitel 3 die Abbildungen 17, 18, 19 und 20 beachtet werden. Der Initialprozess beginnt dadurch, dass ein Kunde im CFM erfasst wird, dies wird dann parallel an die Finanzbuchhaltung übertragen, wo dieser Kunde dann als buchhalterischer Debitor angelegt werden muss.

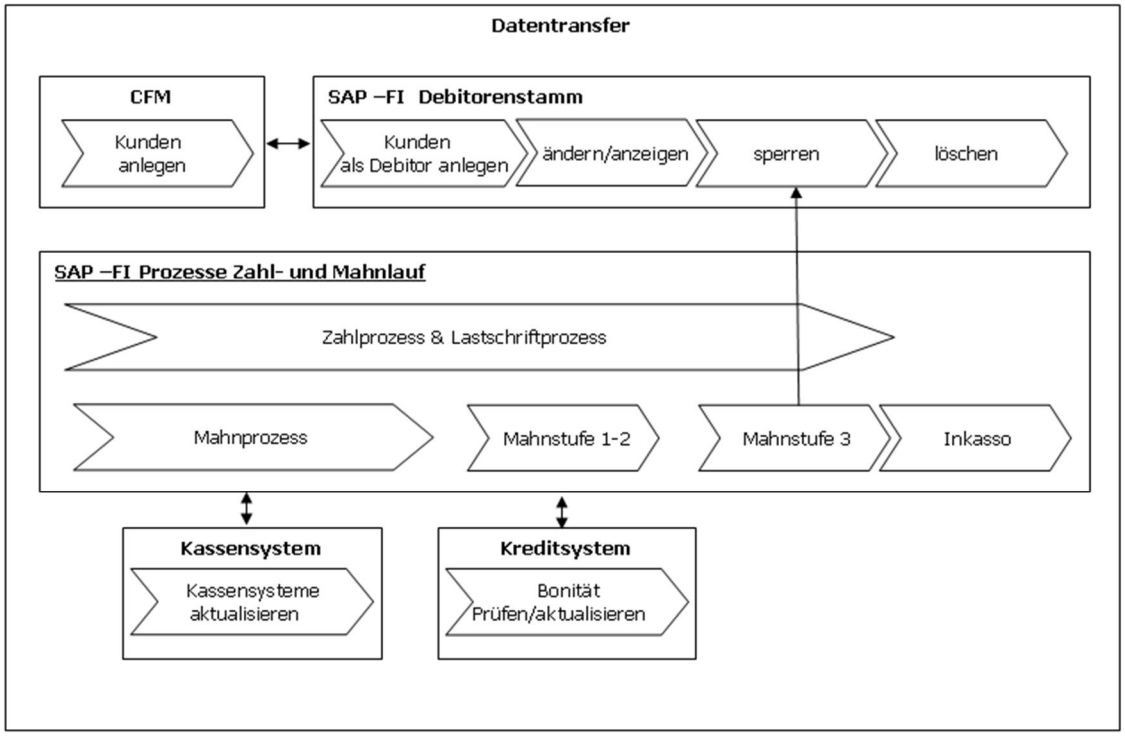


Abbildung 74: SAP-FI-Prozess und die Auswirkungen auf angebundene Systeme

4.3.1 Stammdaten vom CFM zum SAP-FI

Für die Qualität und die Korrektheit der Debitorenstammdaten sind die Datenersteller aus den jeweiligen Vorsystemen[95], wie das CFM, verantwortlich [33]. Die im CFM angelegten Stammdaten eines Kunden können jetzt nach der Übertragung ins SAP durch SAP-Transaktionen angezeigt und um weitere buchhalterische Daten ergänzt werden [83].

CFM übergibt einen Datensatz mit einer Kundennummer, SAP-FI generiert aus der Kundennummer zusätzlich noch eine Debitorennummer, so ist der Kunde auch in der Buchhaltung mit Hilfe der Debitorennummer zu identifizieren.

[95] Softwaresysteme, die Daten zu SAP-FI liefern.

4.3.1.1 Testfall: Stammdaten anlegen, ändern und anzeigen (Debitoren) im SAP-FI

Die beiden Prozessschritte des Anlegens und des Änderns von Debitoren können zusammengefasst werden, da diese über eine einheitliche SAP-Transaktion (FD03) aufgerufen werden.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Vorbedingung	SAP-Applikationsserver installiert und konfiguriert, Aufruf von SAP-Anmeldemaske über SAP Logon	Anmeldemaske erscheint
Schritt 52	Eingeben der Daten: SAP System >> ? << [String] SAP Client >> ? << [String] Benutzername >> ? << [String] Ländersprache >> ? << [String] Passwort >> ? << [String] klick auf „Enter“	SAP-FI-Einstiegsbildschirm erscheint
Schritt 53	Aufrufen der Transaktion „FD03“	Debitoren anzeigen, Einstiegsbildschirm erscheint
Schritt 54	Debitorennummer >> ? << [String] eingeben	Allgemeine Daten, Einstiegsbild
Schritt 55	(anlegen, ändern) von Daten – „ausführen“	Datensatz erfolgreich verändert

4.3.1.2 Testfall: Stammdaten löschen

Wenn Debitorenstammdaten aus dem System gelöscht werden sollen, muss dafür im SAP ein Löschkennzeichen gesetzt werden [142]. Da hier gesetzliche Vorgaben zu beachten sind, kann der Debitor nicht unmittelbar aus dem System gelöscht werden, sondern er muss noch eine Zeit[96] im System verweilen.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Vorbedingung	SAP-Applikationsserver installiert und konfiguriert. Aufruf von SAP-Anmeldemaske über SAP Logon	Anmeldemaske erscheint
Schritt 56	Eingeben der Daten: SAP System >> ? << [String] SAP Client >> ? << [String] Benutzername >> ? << [String]	SAP-FI-Einstiegsbild-schirm erscheint

[96] Es gibt für verschiedene Prozesse und Anwendungsfälle unterschiedliche Löschfristen.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
	Ländersprache >> ? << [String] Passwort >> ? << [String] klick auf „Enter“	
Schritt 57	Aufrufen der Transaktion „FD06“	Debitoren-Löschvermerk anzeigen, Einstiegsbild-schirm erscheint
Schritt 58	Debitorennummer >> ? << [String] eingeben	Allgemeine Daten, Einstiegsbild
Schritt 59	Löschvermerk setzen, dann „ausführen“	Datensatz erfolgreich verändert

Eine weitere Facette bei den Debitorenstammdaten ist das Sperren von Debitoren. Dieser Prozess kommt immer dann in Betracht, wenn Mahnprozesse keinen Erfolg gebracht haben. Bleiben die Zahlungen auch beim dritten Mahnversuch aus, so müssen säumige Zahler zum Schutz des Unternehmens, aber auch anderer Geschäftspartner, gesperrt werden. Ein Kauf von Artikeln ist dann nur noch gegen Barzahlung möglich. Sowohl der Mahn- als auch der Sperrprozess erfolgen in vielen Unternehmen nach demselben Schema.

4.3.1.3 Testfall: Stammdaten sperren

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Vorbedingung	Der Zahlprozess wurde nicht erfolgreich durchlaufen und scheiterte an einem nicht gedeckten Kundenkonto. Kunden-Mahnprozess scheitert auch im dritten Mahnlauf.	
Vorbedingung	SAP-Applikationsserver installiert und konfiguriert. Aufruf von SAP-Anmeldemaske über SAP Logon	Anmeldemaske erscheint
Schritt 60	Eingeben der Daten: SAP System >> ? << [String] SAP Client >> ? << [String] Benutzername >> ? << [String] Ländersprache >> ? << [String] Passwort >> ? << [String]	SAP-FI-Einstiegsbildschirm erscheint

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
	klick auf „Enter“	
Schritt 61	Aufrufen der Transaktion „FD05“	Debitoren Entsperren/Sperren anzeigen Einstiegsbildschirm erscheint
Schritt 62	Debitorennummer >> ? << [String] eingeben – Sperren – „sichern“	Kunde wird gesperrt

Der Prozess des Sperrens bzw. Entsperrens eines Kunden erfolgt in der Praxiswelt automatisiert. Bringen sowohl der Zahl- als auch der Mahnprozess keinen Erfolg, so greifen „Automatismen“ aus dem SAP, was dazu führen kann, dass Kunden gesperrt werden. Dieser Prozess muss immer getestet werden, da Änderungen dazu führen können, dass genau dieser Automatismus zu Fehlern führt, wie das Sperren von Kunden trotz Zahlung aller offenen Rechnungen.

Der Debitor ist jetzt sowohl im Kundenmanagementsystem als Kunde mit einer Kundennummer aufgeführt als auch in der Buchhaltung mit einer Debitorennummer versehen. Im nächsten Schritt müssen die zu den Debitoren gehörenden Umsätze, die offenen Posten oder auch die bezahlten Einkäufe über den Kassenserver zu SAP-FI übertragen werden.

4.3.2 Schnittstelle zum Kassenserver

Bei der Kassenabrechnung handelt es sich um die Übergabe der täglichen Kasseneinnahmen in den Märkten sowohl von Barverkäufen, EC- und Kreditkartenverkäufen als auch von Rechnungsverkäufen mit Zahlungsziel an den Kunden (Lastschrift). Diese Daten werden täglich über eine Schnittstelle vom Kassenserver ins SAP übertragen. Die über die Schnittstelle bereitgestellten Informationen werden noch am gleichen Tag durch SAP aufbereitet. Kreditkäufe oder Lastschriftbezahlungen (sind Krediteinkäufe) werden direkt auf die Debitorenkonten gebucht, damit die Berechnung des Kreditlimits noch vor der nächsten Marktöffnung erfolgen kann.

Das Kassensystem verwendet dazu Transaktionstypen und Sequenznummern, um die Art des Geschäftsvorfalls zu bestimmen. Diese Transaktionstypen und Sequenznummern werden anschließend von der Schnittstelle identifiziert, um die korrekte Buchung in den SAP-FI-Konten zu verwenden. Z. B wird für das Einkaufen per Kreditkarte als Transaktionstyp „600“ als Parameter verwendet, so ist SAP-FI in der Lage, zu bestimmen, wie dieser von der Kasse gemeldete offene Posten buchhalterisch im FI zu verbuchen ist.

4.3.2.1 Testfall-Daten, die vom Kassensystem zu SAP-FI übertragen werden

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Vorbedingung 1	Kassensystem-Tagesabschluss abgeschlossen und an SAP-FI übertragen	
Schritt 63	Kundennummer >> ? << [String] in das Kassensystem eingeben	Auf dem Kassendisplay werden alle Kundeninfor-mationen angezeigt
Schritt 64	Erfassen aller Waren mit einem Scanner, an dieser Stelle können Waren und Produkte eingescannt werden	Alle erfassten Artikel werden auf dem Kassendisplay angezeigt
Schritt 65	Zahlungsvariante wählen [bar] [Lastschrift, z. B. EC-Karte] [Kreditkarte] [Rechnung] [Schecks] [Gutscheine]	
Schritt 66	Je nach Zahlungsvariante kann der Prozess verschiedene Pfade durchlaufen	Sobald die Kasse diesen Verkauf als erledigt markiert, werden die Daten zum späteren Transferieren auf dem Kassenserver abgelegt
Vorbedingung 2	Um einen Tagesabschluss am Kassensystem zu starten, muss sich der Hauptkassierer oder jemand, der die Rolle des Hauptkassierers annehmen darf, am Kassensystem anmelden.	
Schritt 67	Tagesabschluss am Kassensystem durchführen, „Tagesabschluss“ klicken	Der Tagesabschlusslauf kann einige Minuten benötigen, so werden alle an dieser Kasse und im Markt gemachten Umsätze am Tag ermittelt, berechnet und direkt ins FI transferiert

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 68	Die an der Kasse und im Markt verarbeiteten Umsätze werden an SAP zur Aufbereitung weitergeleitet	Daten sind in das SAP übertragen und werden automatisch in den SAP-Tabellen für Kassenumsätze gespeichert

4.3.2.2 Testfall-SAP-FI: Einlesen der Kassendaten

Voraussetzung für das Einlesen der täglichen Kassenabrechnung ist die erfolgte Bereitstellung der Tagesdaten auf dem SAP-FI-Server (Testschritte 67 und 68).

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Vorbedingung	Testschritte 67 und 68 erfolgreich abgeschlossen.	
Schritt 69	SAP-Applikationsserver installiert und konfiguriert. Aufruf von SAP-Anmeldemaske über SAP Logon	Anmeldemaske erscheint
Schritt 70	Eingeben der Daten: SAP System >> ? << [String] SAP Client >> ? << [String] Benutzername >> ? << [String] Ländersprache >> ? << [String] Passwort >> ? << [String] klick auf „Enter“	SAP-FI-Einstiegsbildschirm erscheint
Schritt 71	Aufrufen der Transaktion „YAR03_STORE“	Vorhandensein von Debitoreneinzelposten, diese müssen über die Kassensystem-schnittstelle verfügbar sein
Schritt 72	Transaktion „YAR03_STORE“ ausführen und Felder: Buchungskreis >> ? << [String] Kostenstelle >> ? << [String] Abschlusstag >> ? << [Date] ausfüllen, dann auf „ausführen“ klicken	Die mittels der Kassenschnittstelle übertragene Datei wird jetzt im SAP-System eingebucht.
Schritt 73	Reiter „Abstimmung“ im SAP „anklicken“	Prüfen, ob die

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
		Abstimmung ohne Fehler durchgelaufen ist. Ein typischer Fehler kann das falsche Verbuchen einer Tagesabschluss-Datei aus der Kasse sein oder auch eine abgebrochene Transaktion während des Buchungslaufs im SAP.
Schritt 74	Sind die Abstimmungen ohne Fehler, muss der Tagesabschluss eingebucht werden. Transaktion „FB03“	Verbuchung im SAP erfolgreich, Meldung „Tagesabschluss buchen“ erscheint in der Statusleiste.

Unter Schritt 74 wird im SAP-System ein Beleg erzeugt, welcher dann durch einen Buchhalter geprüft werden muss. Jetzt erfolgt die Weiterverarbeitung im SAP.

4.3.2.3 Testfall: Weiterverarbeitung der Kassendaten im SAP-FI

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Vorbedingung	Testschritte 72–74 erfolgreich abgeschlossen. Beleg erzeugt.	
Schritt 75	Transaktion „SE38“ im SAP aufrufen	Aufforderung erscheint, welches Programm oder welcher Report ausgeführt werden soll
Schritt 76	Transaktion: Y00 eingeben und ausführen	Offene Posten zu allen Kunden werden angezeigt, mittels einer Debitorennummer >> ? << [String] kann dann nach einem Kunden selektiert werden
Schritt 77	Transaktion „Y00“,	Selektieren durch

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
	damit werden alle offenen Posten zu dem Kunden angezeigt	Eingrenzen in der Liste mittels Debitorennummer.

Mit dem Ausführen der Transaktion (Y0BAL) [33] kann jetzt die Zahlungsvorschlagsliste erstellt werden, welche die Basis für den Zahlprozess bildet. Die Zahlungsvorschlagsliste wird durch SAP angefertigt und ermittelt alle Debitoren, welche ihre Zahlungsfrist erreicht haben.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 78	Transaktion „F110“	Programm zur Erstellung eines Zahlungsvorschlags
Schritt 79	Kopfdaten zum Zahlungsvorschlag eingeben: Tag der Ausführung >> ? << [Date] Identifikation >> ? << [String] Buchungsdatum >> ? << [Date] Belege erfasst bis >> ? << [Date] Debitoren fällig bis >> ? << [Date] Buchungskreis >> ? << [String] Zahlweg >> ? << [String] Buchungsdatum >> ? << [Date] Debitor >> ? << [String] (einzeln, Gruppe) „ausführen“ klicken	Liste zum Zahlungsvorschlag zu den selektierten Debitoren wird ermittelt. Und ein SAP-Fenster „Zahlung einplanen“ erscheint.
Schritt 80	Im nächsten Schritt wird jetzt der Zahllauf geplant, dazu werden Startdatum >> ? << [Date] Startzeit >> ? << [Time] eingegeben	Zahlprozess startet zum angegebenen Datum. Werden die Felder leer gelassen, wird immer das aktuelle Tagesdatum als Startdatum und die aktuelle Zeit als Startzeit gewählt.

4.3.3 Zahllauf

Bevor der Testablauf für den automatischen Zahllauf gestartet wird, sollten zunächst die für den jeweiligen Buchungskreis gültigen Zahlwege und die Stammdateneinstellungen der zu regulierenden Debitoren betrachtet werden (Testschritte 78–80). Je nach verwendetem Zahlweg im Buchungskreis ergeben sich unterschiedliche Abläufe, die durch die jeweiligen Tests nachvollzogen und bestätigt werden sollen. Nach erfolgter Verarbeitung des Zah-

lungsvorschlags kann die Zahlliste vor der Einplanung des eigentlichen Zahllaufes noch bearbeitet werden, um z. B. einzelne offene Posten von der Zahlung auszunehmen.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 81	Transaktion „F110“	Programm zur Erstellung eines Zahlungsvorschlags
Schritt 82	Führe Testschritte 79–80 aus. Nach Testschritt 80 Programm ausführen, so startet jetzt der Zahlprozess	Zahlungsvorschlags-liste wird aus dem SAP generiert. Einzelposten werden in einem Beleg aufgelistet.
Schritt 83	Transaktion „FBL1N“ aufrufen	Überprüfung der gebuchten Zahlungen von Kundenkonten auf Unternehmenskonten
Schritt 84	Debitorennummer >> ? << [String] Zuordnung >> ? << [String] Belegnummer >> ? << [String] Belegart >> ? << [String] Belegdatum >> ? << [Date] Betrag >> ? << [String] Währung >> ? << [String]	Die vom Kassensystem gemeldeten offenen Posten wurden erfolgreich in die Zahlungsvorschlags-liste von SAP übernommen, anschließend durch den Zahlprozess geführt und die Beträge konnten vertragsgemäß vom Kundenkonto abgebucht werden

4.3.4 Bankenmanagementsystem

Viele Unternehmen im B2B wickeln ihre Finanztransaktionen über eigene Hausbanken ab [142], [33]. SAP bietet im Rahmen des SAP-FI-Moduls spezielle Bankenapplikationen an, welche in der Lage sind, die finanziellen Transaktionen durchzuführen. Aus dem Zahllauf wird eine Zahllaufliste erstellt, welche an die Bankenapplikation von SAP übergeben wird.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 85	Testschritt 79–80 ausführen	Liste mit Zahlungsvorschlag

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
		wird erzeugt
Schritt 86	Transaktion „SE38“ aufrufen	Aufforderung erscheint, welches Programm oder welcher Report ausgeführt werden soll
Schritt 87	Transaktion „YSDI[97]“ aufrufen	Liste aller verfügbaren Schnittstellen – Bankenschnittstelle selektieren
Schritt 88	Transaktion YP[98] aufrufen	„Datensatznummer anlegen“ erscheint
Schritt 89	Hauptbuchungskreis >> ? << [String] SDI-Schnittstelle MGP Datei: >> ? << [String] ausführen	Einlesen der Zahllaufliste
Schritt 90	Transaktion „SM35“	„SAP Batch Input Mappe“-Auswahlliste
Schritt 91	Transaktion „APPAY“	Batch-Input-Mappe wird ausgeführt und die Zahlungstransaktionen werden durchgeführt. Kontoauszug wird generiert.
Schritt 92	Transaktion „FB03“	Belege anzeigen – vergleichen
Schritt 93	Elektronischen Kontoauszug einlesen mit Hilfe der SAP Standardtransaktion: „FF_5“	Alle offenen Posten in SAP-FI zu einem Kunden (wenn ein

[97] Serial Document Interface.
[98] Je nach Unternehmen kann das YP im Transaktionsnamen (YP) ersetzt werden – im Fallbeispiel steht das Y als Standardkürzel im SAP für eigens entwickelte Programme, P steht für Payment.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
		Lastschrifteinzug erfolgreich war) werden ausgeglichen. Kreditlimit neu berechnen.

Die erfolgreich ausgeführten Transaktionen führen jetzt dazu, dass alle gemeldeten offenen Posten nach Schritt 91 ausgeglichen sein müssen. Die Beträge, welche von Kundenkonten per Lastschrift eingezogen worden sind, müssen in die Debitorenbuchhaltung zurückgemeldet werden. Gleichzeitig werden die Kreditlimits der Kunden nach erfolgreicher Zahlung aktualisiert.

Dazu wird mittels SAP eine Datei generiert und an die angebundenen Systeme (CFM, Kasse und Kreditmanagementsystem) übergeleitet. Dies beinhaltet alle offenen und ausgeglichenen Posten eines Kunden. Ist allerdings der Zahlprozess bei einigen Kunden nicht erfolgreich, so muss ein Mahnprozess angestoßen werden, der dafür sorgen soll, den Kunden an die offenen Rechnungsbeträge zu erinnern.

4.3.5 Mahnlauf Stufe 1

Bevor der Testablauf für den Mahnprozess gestartet wird, sollten zunächst das für den jeweiligen Buchungskreis gültige Mahnverfahren und die Stammdateneinstellungen der zu mahnenden Kunden betrachtet werden. Abhängig von der im Kundenstamm aktualisierten Mahnstufe können sich bei den am Test beteiligten Kunden unterschiedliche Verarbeitungen beim nachstehend durchzuführenden Mahnlauf [33] ergeben. Der Mahnlaufprozess muss parallel zum Zahllaufprozess angelegt werden, da dieser Prozess in der Praxis dann automatisiert mit dem Zahllauf gestartet wird.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Vorbedingungen	Testschritt 81–93 teils erfolgreich ausgeführt	
Schritt 94	Transaktion „F150“ ausführen	Liste mit Zahlungsvorschlägen, die auf Fehler gelaufen sind, wird erzeugt
Schritt 95	Anlegen des Mahnvorschlags auch über die Transaktion „F150“	
Schritt 96	Ausführung am >> ? << [Date] Identifikation >> ? << [String] Mahndatum>> ? << [Date] Buchungskreis >> ? << [String] Debitoren >> ? << [String]	Festlegung, an welchem Tag der Mahnlauf gestartet werden muss, Angabe der Buchungskreise für die der Mahnlauf durchgeführt werden

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
		soll, Liste der Debitoren, die angemahnt werden müssen
Schritt 97	Mahnlauf „sichern“	Einplanungsprogramm startet

Nachdem der Mahnlauf gesichert ist, muss die Einplanung für diesen Mahnlauf vorgenommen werden. Da der Mahnlaufprozess immer parallel zum Zahllaufprozess konfiguriert werden muss, können die selektierten Debitoren mit dem Zielzahlungsdatum genommen werden, und sollte nach dem Zielzahlungsdatum keine erfolgreiche Abarbeitung des Zahlprozesses erfolgt sein, dann soll automatisch der soeben angelegte Mahnlauf gestartet werden.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Schritt 98	Die Planung des Mahnlaufs kann über die Transaktion „F150“ vorgenommen werden.	
Schritt 99	Transaktion „F150“ ausführen	Selektion und Druck des Mahnlaufs planen
Schritt 100	Startdatum >> ? << [Date] Startzeit >> ? << [Time] Mahndruck >> ? << [String] Ausgabegeräte >> ? << [String] (Liste von Druckern) und auf „einplanen“ klicken	Mahnlauf gesichert
Schritt 101	Nach erfolgter Verarbeitung des Mahnlaufes kann die Mahnliste vor der Einplanung des Mahndruckes noch bearbeitet werden, um z. B. einzelne offene Posten von der Mahnung auszunehmen	
Schritt 102	Der Mahnlauf kann jetzt automatisch durch den Zahllauf gestartet werden, sobald dieser die offenen Posten nicht ausgleichen kann	
Schritt 103	Sobald der Mahnlauf einmal gestartet worden ist, kann zu Testzwecken mittels der SAP-Transaktion „FD03“ jetzt überprüft werden, ob die Mahnstufe nach oben korrigiert werden konnte	Debitorennummer und Parameter Mahnstufe muss auf 1 stehen.
104	Unmittelbar nachdem der Mahnlauf gestartet	1. Mahnung,

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
	worden ist, wird automatisch mit der Transaktion „SP01“ der Druck des Mahnbriefs eingeleitet	Briefkopf mit Kundendaten und Mahnbetrag
105	Transaktion „F110“	Programm zur Erstellung eines Zahlungsvorschlags
106	Führe Testschritte 79–80 aus. Nach Testschritt 80 Programm ausführen, so startet jetzt der Zahlprozess.	Zahlungsvorschlags-liste wird mittels SAP generiert. Einzelposten werden in einem Beleg aufgelistet.
107	Der Zahllauf zum angemahnten Debitor wird wieder für den nächsten Zahlungslauf nach der gesetzlichen Wartefrist eingeplant	Debitor wird in der Zahlungsvorschlags-liste aufgeführt, mit dem Vermerk, dass dieser die erste Mahnstufe erreicht hat

Um das Mahnverfahren vollständig zu testen, sollten in Abhängigkeit vom Mahnverfahren alle Mahnstufen durchgetestet werden. Hierbei ist der Mahnabstand im jeweiligen Mahnverfahren zu beachten. Dieser wird vom SAP-System vorgegeben und kann evtl. durch Sachbearbeiter vom jeweiligen Fachbereich angepasst werden. Für den Test werden alle Debitoren ausgesucht, die bereits eine erhöhte Mahnstufe erreicht haben, mit Ausnahme der letzten Mahnstufe. Der Unterschied ergibt sich im Ablauf aus den Mahnbriefen und nach Abhängigkeit vom Mahnverfahren in der Übergabe ans gesetzliche Mahnverfahren bzw. an einen Inkassobeauftragten, der sich um die Zahlung der überfälligen Beträge bemüht. Auch muss beachtet werden, ob Verzugszinsen richtig berechnet werden, wenn dies im Mahnverfahren vorgesehen ist. Zinsen werden aber nur in Abhängigkeit vom Eintrag im Stammsatz des Debitors vorgenommen. Ebenfalls muss geprüft werden, ob Mahngebühren richtig verrechnet werden.

4.3.6 Mahnlauf Stufe 2 und 3

Die zweite Mahnstufe wird eingeleitet, nachdem die erste Mahnstufe zu keinem Erfolg geführt hat.

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
Vorbedingungen	Testschritt 81–93 teils erfolgreich ausgeführt	
Vorbedingungen	Prüfen der offenen Posten des Debitors, ob diese tatsächlich noch offen sind	Offene Posten sind nicht ausgeglichen

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
	Transaktion „FD03“ Debitorennummer >> ? << [String]	
Schritt 108	Transaktion „F150“ ausführen	Liste mit Zahlungsvorschlägen, die auf Fehler gelaufen sind, wird erzeugt
Schritt 109	Anlegen des Mahnvorschlags auch über die Transaktion „F150“	
Schritt 110	Ausführung am >> ? << [Date] Identifikation >> ? << [String] Mahndatum>> ? << [Date] Buchungskreis >> ? << [String] Debitoren >> ? << [String]	Festlegung, an welchem Tag der Mahnlauf gestartet werden muss, Angabe der Buchungskreise, für die der Mahnlauf durchgeführt werden soll, Liste der Debitoren, die angemahnt werden müssen.
Schritt 111	Mahnlauf „sichern“	Einplanungsprogramm startet
Schritt 112	Mahnprozess nach gesetzlicher Wartefrist im Zahlprozess starten. Bedingung: Debitor konnte im Schritt 81–93 die offenen Posten nicht ausgleichen.	
Schritt 113	Mahnstufe 2 gestartet Startdatum >> ? << [Date] Startzeit >> ? << [Time] Mahndruck >> ? << [String] Ausgabegeräte >> ? << [String] (Liste von Druckern)	
Schritt 114	Datei mit offenen Posten zu den angemahnten Debitoren wird generiert und automatisch durch SAP-FI an die Kassensysteme, an das Kreditmanagementsystem und an das CFM versendet	Die von SAP-FI übermittelten Daten zu einem säumigen Debitor werden in das Kassensystem und das Kreditsystem

Aktion	Beschreibung Testschritt	Erwartetes Ergebnis
		transferiert
Schritt 115	Login in das Kassensystem, Prüfen der Kreditlimits, Login in das Kreditmanagementsystem, Prüfen der Kreditlimits, Login-CFM prüfen, das Kreditlimit prüfen	Übereinstimmung der Kreditlimits mit dem im SAP-System
Schritt 116	Informieren der Geschäftspartner über Zahlungsschwierigkeiten eines Debitors über die externen Wirtschaftsauskunfteien	Die von SAP-FI an das Kreditmanagement-system überlieferte Datei enthält Informationen wie: Mahnstufe 2 erreicht. Dies wird vom Kreditmanagement-system verarbeitet

Führt auch eine zweite Mahnstufe zu keinem Erfolg, so wird eine dritte Mahnstufe direkt durch ein Inkassounternehmen übernommen, dazu wird in SAP-FI der Debitor für Käufe auf Zeit gesperrt. Analog dazu wird sein Kreditlimit eingefroren und externe Wirtschafts-auskunfteien werden direkt vor diesem Debitor gewarnt. Dieser Prozess erfolgt in der Praxis automatisiert. Deshalb muss parallel zum Anlegen eines Zahlungslaufs direkt auch der Mahnlaufprozess bis zur dritten Mahnstufe konfiguriert werden. So ist sichergestellt, dass die Abläufe IT-gestützt erfolgen und dass dieser Kunde gemahnt wird, solange kein Zahlungsausgleich zu einem offenen Posten erfolgt.

Die hier beschriebenen Testspezifikationen sind in den Testskripten, welche nach der Adaptierung entstehen, abgebildet.

4.4 Zusammenfassung und Analyse der Ergebnisse aus den Kapiteln 3 und 4

Der Zweck der in dieser Arbeit vorgestellten Methode der modellbasierten Testfallgenerierung und der anschließenden Adaptierung an fertig ausführbare Testskripte ist es, implizit sowohl die textuell schon vorhandenen beschriebenen Testfälle als auch die formalisierten Testanforderungen in den generierten Testfällen stärker zu fokussieren, als dies bisher mit Techniken zur Testfallermittlung möglich ist. Das Ziel ist es vor allem, einen Stamm an Regressionstests zu entwickeln, welcher den Tagesbetrieb der geschäftskritischen Prozesse (am Fallbeispiel des IT-gestützten Zahlungsabwicklungsprozesses) nach Erweiterungen oder Anpassungen in den Softwaresystemen sicherstellt [91].

Insbesondere sollen die Gewichtungen der Testschritte und Pfade in einem Testmodell ermöglicht werden. Somit lassen sich Testfallgenerierungen nach Kritikalität steuern. Das Ziel der Priorisierung nach Kritikalität ist nicht nur die Bevorzugung höher priorisierter

Testfälle, sondern damit rücken die Prozesse in den Vordergrund, welche eine Prozessüberdeckung ermöglichen. Gleichzeitig versucht die Methode, die Qualität der generierten Testfälle und Testskripte in einer frühen Entwicklungsphase positiv zu beeinflussen. Bestenfalls werden Fehler schon bei der Modellierung der Testfälle identifiziert und behoben.

Bei der Modellierung werden besonders fehlerbehaftete Teile des zu testenden Systems vorrangig im Test berücksichtigt. Die Modellierung erfolgt aufgrund von Erfahrungswerten aus Fachbereichen und Testergebnissen früherer Testabläufe. Spezifizierte Testszenarien sind exemplarische Repräsentanten von Äquivalenzklassen, welche auf der Basis der Testmodelle und Testdaten jeweils zusätzlich gebildet werden müssen.

Der größte Vorteil des MBT ist die Verwendung eines sowohl mit dem Fachbereich als auch mit der Entwicklung abgestimmten Testmodells (Systemverhaltensmodell). Dieses ermöglicht es, dass sowohl die Anforderungssteller (Fachbereiche) als auch Entwickler und Tester gemeinsam miteinander auf Basis dieses Testmodells kommunizieren können, wodurch ein einheitliches Verständnis erzeugt wird. Auch bietet das Testmodell den weiteren Vorteil, dass anhand des Modells eine Testfallmenge systematisch erzeugt werden kann. Nur durch systematisches Vorgehen können Fehler frühzeitig identifiziert und behoben werden. Gleichzeitig bietet das systematische Vorgehen ein Kriterium, mit welchem ein Ende einer Testphase definiert werden kann. So kann eine Testphase als beendet gewertet werden, wenn ein Überdeckungskriterium erfüllt ist.

4.4.1 Beobachtungen: Anforderungsüberdeckung und Prozessüberdeckung

Die vorgestellte Methode führt dazu, dass sich Testfälle stärker an der Anforderungsspezifikation orientieren. Dies wird durch mehrere Beobachtungen belegt: Zunächst zeigt bereits die Analyse der gewonnenen BPMN-Testmodelle, dass die höher gewichteten Testanforderungen stärkere Berücksichtigung finden (Zahlprozess, Mahnprozess usw.). Das schlägt sich dadurch nieder, dass diese Testschritte mit der Priorität 1 häufiger im Testmodell zu finden sind. Die Transitionen, welche von diesen Szenarien ausgeführt werden, kommen überproportional häufig vor. Weiter wird durch die Analyse der Testfälle deutlich, dass diese Transitionen auch vermehrt in den Testfällen ausgeführt werden, welche aus den Teilmodellen gewonnen wurden. Im Gegensatz dazu werden die Transitionen, welche für die Ausführung der Anforderungen charakteristisch sind, bei Testsuiten vor dem Hintergrund von rein textuell erfassten Testfällen auf der Basis von Testfallspezifikationen weniger häufig durchlaufen.

Bei der Anforderungsüberdeckung bestätigt sich die Bewertung, dass Testfälle, welche mit dieser vorgeschlagenen Methode ermittelt werden, stärker an die Anforderungsspezifikation angelehnt sind als die, welche textuell aus Testfallspezifikationen oder Anforderungsspezifikationen gewonnen wurden.

Die Bewertung der Anforderungsüberdeckung auf der Basis von generierten Testfällen nimmt dabei an, dass die Testfälle exemplarische Repräsentanten von Anforderungen sind und damit Anforderungen hinsichtlich des Auftretens von Eingabeaktivitäten, hinsichtlich der Reihenfolge dieser Aktivitäten und hinsichtlich der Sequenzen von Aktivitäten be-

schreiben. Damit können Bedingungen identifiziert werden, unter denen ein Testfall eine Anforderung prüft und somit zu einer Überdeckung des Szenarios beiträgt.

Bei der Prozessüberdeckung ist es nicht zwingend notwendig, jeden modellierten Pfad im BPMN-Testmodell zu navigieren und daraus Testfälle abzuleiten. Vielmehr führt die Priorisierung der wichtigsten Pfade im Testmodell dazu, eine Aussage darüber zu treffen, ob eine Prozessüberdeckung erreicht werden konnte oder nicht. Wichtig für die Prozessüberdeckung ist es, tatsächlich die für sowohl den Geschäfts- als auch für den Testprozess notwendigen Transaktionen durchzuführen. Eine Einschätzung, ob eine Prozessüberdeckung vollständig ist, kann anhand der im Softwaresystem eingesetzten Transaktionen errechnet werden. So können Auswertungen über die wichtigsten Transaktionen in einem Softwaresystem durchgeführt und mit den genutzten Transaktionen im Test verglichen werden. Im Test müssen sich alle genutzten geschäftskritischen Transaktionen wiederfinden. So lässt sich die Prozessüberdeckung messen.

Im hier vorgestellten IT-gestützten Zahlungsabwicklungsprozess als Fallbeispiel wurden alle geschäftskritischen Transaktionen berücksichtigt und sind somit Gegenstand des modellbasierten Testens.

4.4.2 Analyse der Ergebnisse

Während der im Rahmen dieser Dissertation durchgeführten Interviews sowohl mit externen als auch mit internen Partnern[99], Kollegen und Fachbereichen wird das Thema Testen nach der Vorstellung der Methode aus einem neuen Blickwinkel betrachtet. An den Interviews beteiligten sich insgesamt 12 Testmanager von insgesamt acht befragten Unternehmen (siehe dazu auch Kapitel 2.4, 2.4.1 und 2.4.2). Der dazu herangezogene Fragenkatalog befindet sich im Anhang dieser Dissertation. Anhand dieser Fragen führte der Autor dieser Dissertation dann die Interviews durch, welche in Kapitel 2.4 ausgewertet wurden.

Die Auswertung erfolgte anhand der Ergebnisse (Antworten der Testmanager), welche der Autor während der Befragung protokollierte. Die Auswertungen wurden sinngemäß und mit den von den Interviewpartnern jeweils angegebenen Zahlen wiedergegeben.

Durch die Interviews konnte eine Einschätzung gewonnen werden, inwiefern andere Unternehmen modellbasierte Testtechnologien einsetzen, und es war interessant, zu erfahren, wie die Unternehmensorganisation des jeweiligen Unternehmens solche Testwerkzeuge unterstützt. Auch was MBT in diesen Organisationen an Mehrwert oder Nachteilen impliziert, wurde schon im Kapitel 2.4 zusammengefasst.

Der Kernteil dieser Dissertation demonstriert, wie aus der Mischung aus klassischen und modellbasierten Methoden eine Testmethode entstand, die sowohl das Berufsbild des Testers als auch die Fachbereiche verändert. Ein Tester oder Business-Analyst muss entsprechend früh in einen Entwicklungsprozess eingebunden werden. Seine Aufgabe ist es, Testanforderungen in Testmodelle zu überführen, diese zu priorisieren und zusammen mit den Fachbereichen die Modelle dabei gleichzeitig einem Review[100] zu unterziehen. Die Fachbereiche integrieren sich mehr in die Arbeit der Business-Analysten und sammeln erste Er-

[99] Verschiedene Gesellschaften im Konzern.

[100] Formale Prüfung auf Korrektheit.

fahrungen mit der Modellierung von Testprozessen. Da in betrieblichen Informationssystemen zunehmend BPMN als Modellierungsnotation in Fachbereichen eingesetzt wird [151], bietet es sich an, die Fachbereiche an die Modellierungsnotation heranzubringen. Der Fokus liegt ausschließlich auf der Entwicklung robuster Regressionstests, welche sukzessive in Zusammenarbeit mit Fachbereichen und IT-Experten erweitert werden.

4.4.3 Funktionen und Testskripte als Bausteine

Die entwickelten Testskripte und Funktionsbibliotheken wurden generisch programmiert und sind wiederverwendbar. So können die Funktionen in den Bibliotheken als Bausteine interpretiert und nach einem Baukastenprinzip zusammengestellt werden. Die Testlogik ist in Bibliotheken hinterlegt, die Funktionen können dann nach einer vorgegebenen Reihenfolge aus dem jeweiligen Testskript aufgerufen werden. So wird ein Geschäftsprozess in kleine Teile gestückelt und jedes Stück trägt dazu bei, genau diesen Geschäftsprozess auszuführen.

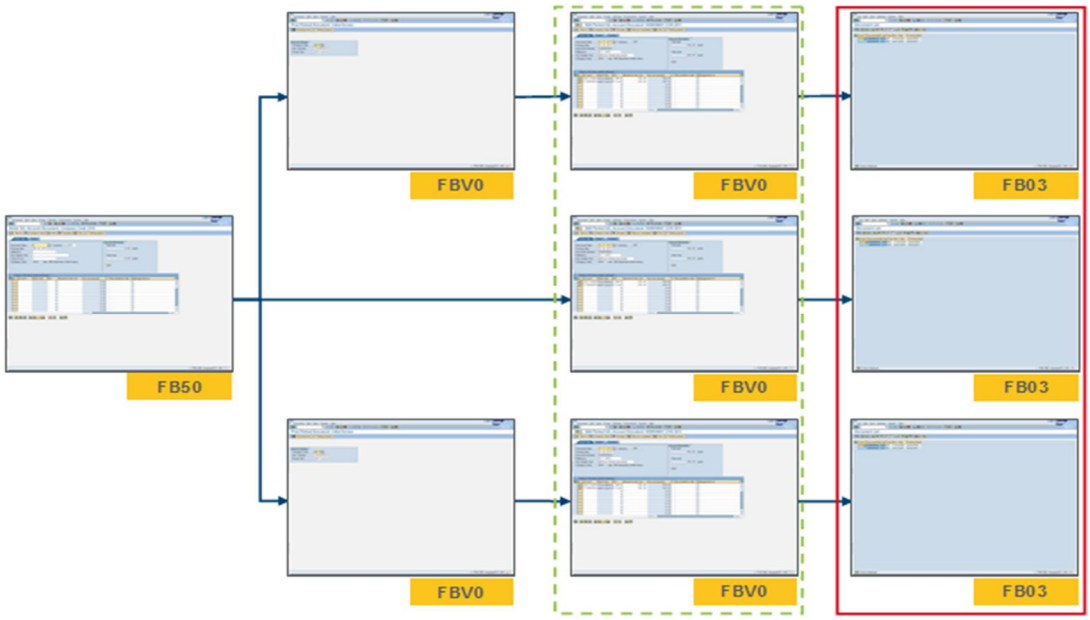


Abbildung 75: Funktionen als wiederverwendbare Bausteine (transaktionsbasiert)

Die farblichen Umrandungen verdeutlichen die Redundanz der Transaktion und gleichzeitig den Vorteil des einmaligen Erstellens durch die generische Programmierung. Die kleinen Teile (Stücke) werden durch Transaktionen repräsentiert, jede Transaktion wird so codiert, dass sie allein durch die für den Test einzusetzenden Testdaten gesteuert wird. So kann im SAP die Transaktion „FB03“ für mehrere verschiedene Buchungsaktionen genutzt werden. Gleichzeitig kann die Transaktion in bspw. 10 verschiedenen Geschäftsprozessen genutzt werden, mit dem Vorteil, dass sie im Testprozess aber nur einmal in VBA codiert ist. Beginnend mit den Standardtestprozessen aller am Fallbeispiel beteiligten Softwaresysteme konnte somit eine feste Basis an Regressionstestfällen angefertigt werden.

Im obigen Beispiel lassen sich die Transaktionen, die hier in den orangefarbenen Kästchen abgebildet sind, so zusammenfügen, wie der Business-Analyst die Testfälle im Testmodell angeordnet hat. Das Beispiel verdeutlicht auch, dass viele Transaktionen in den Prozessen immer wieder benötigt werden, siehe „FB03“ usw. Diese Transaktion wird in allen drei

Pfaden mindestens einmal benötigt, nämlich genau dann, wenn Geschäftsprozesse in den Buchungsmodus gehen und generierte Belege verbucht werden müssen. Zusätzlich lassen sich jetzt noch Prozesse aus den anderen Systemen hinzufügen.

4.4.4 Mögliche Einschränkungen und Seiteneffekte

Bei der Modellierung des CFM-Systems konnte während der ersten Analysen festgestellt werden, dass die bisher durchgeführten Tests zwar gut dokumentiert sind, allerdings bei der näheren visuellen Betrachtung Pfade entdeckt worden sind, welche zuvor in keinem Testkonzept auftauchten. So wurden bisher im Rahmen regressiver Tests die Standard-Transaktionen mit jeweils einer möglichen Testvariante getestet. Der Grund hierfür war bisher die hohe Anzahl an Testfällen, welche ohne Testautomation nicht durchzuführen wären, und die Knappheit von Ressourcen und Zeit.

Durch das erstellte BPMN-Testmodell konnten zusätzliche Pfade für Tests spezifiziert werden, z. B. der Testpfad im CFM, welcher zur Ablehnung eines Kunden führt. In bisherigen Konzepten wurde immer nur der Fall getestet, dass ein Kunde angelegt wird und sich der Bonitätsprüfung unterzieht. Ein weiterer Pfad ist der, bei dem ein Kunde aufgrund einer Sperrung durch säumiges Zahlverhalten oder aufgrund einer negativen Bonitätsabfrage abgelehnt wird. Anhand von Modellen kann jetzt der Blick auf andere Zonen und Komponenten der eingesetzten Software gelenkt werden. Gleiches gilt bei den Kassen- und Kreditsystemen. Durch die Modellierung der Testprozesse konnten Verifikationspunkte eingebaut werden, welche wiederum Testschritte nach ihrem Status abfragen. Dadurch, dass zu jedem Testschritt eine Funktion in der Funktionsbibliothek implementiert wird, können diese Schritte wiederholt ausgeführt werden.

Während der Modellierung der Kernprozesse des Debitoren-Prozesses entstand aus dem Thema heraus ein neues Problem, welches bisher nicht in Betracht gezogen wurde: das Problem der Berechtigungsprüfungen in sensiblen Softwaresystemen. Das Finanzwesen in einem Unternehmen ist ohne Frage ein sehr kritischer und sensibler Bereich. Hier dürfen nur berechtigte Personen Transaktionen durchführen, welche wiederum einem Vier-Augen-Prinzip unterstehen müssen. Das Problem an dieser Stelle war, dass nach jedem Releaseupgrade oder Softwarepatch Benutzer auf einmal berechtigte Rollen im System nicht mehr hatten. Auch wurden alle Kennwörter automatisch neu gesetzt und dies führte zu Fehlern. Während der Bearbeitung dieser Dissertation und der Modellierung der Anmelderoutine in SAP entstand die Idee, die komplette Berechtigungsprüfung im SAP-FI-Modul zu automatisieren.

Die Idee: Nach jedem Releaseupgrade oder auch Softwarepatch, bedingt bspw. durch gesetzliche Änderungen, soll ein Testskript dafür sorgen, dass zu jedem in SAP-FI berechtigten User eine Berechtigungsprüfung durchgeführt wird. Dafür war es notwendig, das Berechtigungskonzept der SAP und der konfigurierten Unternehmensrichtlinie sowie die gesetzlichen Vorgaben zu analysieren und diese in eine Matrix zu überführen. Dabei kam heraus, dass ca. 300 Benutzer das Modul-SAP-FI in der Produktivumgebung täglich nutzen. Im Durchschnitt hat jeder dieser Benutzer 3 Rollen (bspw. Finanzdirektor, Buchhalter, Mitarbeiter in einer Filiale). Jeder dieser Benutzer darf im Durchschnitt ca. 40–45 Transaktionen pro Rolle ausführen. In der Summe sind dies zu viele Transaktionen, um sie manuell

mit einem vertretbaren zeitlichen Aufwand zu testen. Testgegenstand: Transaktionen aufrufen, dann ausprobieren, ob Aktivitäten in den geladenen Transaktionen gestartet werden können, und Transaktion wieder schließen. Alles, was zwischen dem Starten und dem Schließen passiert, sollte lückenlos dokumentiert sein. Genau dieser Prozess wurde im Rahmen dieser Dissertation mit abgewickelt und technisch realisiert. Die entstandenen Funktionsbausteine können auf andere SAP-Module übertragen werden, so kann die hier entwickelte Berechtigungsprüfung auch für andere SAP-Module genutzt werden. Insgesamt testet das Testskript in SAP-FI ca. 44.000 Transaktionsaufrufe und benötigt dafür ca. 22 Stunden. Mit den Transaktionsaufrufen sind auch bewusst Transaktionen hinzugefügt, die ein Benutzer zu Recht nicht haben darf, oder auch eine Anzahl an Transaktionen mit Tippfehlern. Diese dienen dazu, sowohl das Testskript als auch die Güte der Testdaten zu testen. Die festgelegten Namenskonventionen als „Best Practice“-Ansatz bringen einige Vorteile. Nicht nur, dass modellierte Testelemente, Testskripte und Funktionen damit einer einheitlichen Benennung folgen, sondern es kann auch bei der Modellierung der Testfälle ohne eine zusätzliche „Schwimmbahn“ nach der BPMN-Notation ein Element eines anderen Softwaresystems modelliert werden. So kann direkt im Modell ein Übergang zwischen einem Kundensystem und einem Kassensystem dargestellt werden. Bei der anschließenden Testfallgenerierung werden die Testfälle auch genauso abgeleitet.

4.4.5 Probleme und nicht gelöste Anforderungen

Während der Anfertigung dieser Dissertation tauchten Probleme bei der Konfiguration des Modellierungswerkzeugs mit dem Testfallgenerator auf. So mussten explizite BPMN-Testelemente in Form eines Testprofiles durch den Autor dieser Dissertation entwickelt werden, damit der Testfallgenerator diese Elemente auch interpretieren und auswerten kann. Dieser Umstand bot aber auch einen positiven Nebeneffekt. Die Anzahl der BPMN-Elemente konnte auf eine Handvoll reduziert werden. So ist es sowohl für den Business-Analysten als auch für Mitarbeiter der Fachbereiche besonders einfach, Testmodelle zu erstellen. Eine Anforderung, die nicht direkt gelöst werden konnte, ist die Möglichkeit, aus dem Testfallgenerator heraus direkt fertig implementierte und ausführbare Testskripte mit Testlogik zu generieren. Dies hat der Autor dieser Dissertation durch einen Umweg bei der Adaptierung erreicht.

5 Evaluierung der MBT-Methode

Dieses Kapitel enthält die Überprüfung der Aufgabenstellung dieser Dissertation unter Zuhilfenahme von Testwerkzeugen. Dabei wird die entwickelte Methode direkt bei den zu testenden Softwaresystemen angewendet. Damit wird diese Methode gleichzeitig in einer Praxisumgebung evaluiert und auf Praxistauglichkeit untersucht. Wie schon in den Vorkapiteln beschrieben, ist für den Umfang dieser Dissertation das Fallbeispiel der IT-gestützten Zahlungsabwicklung (Debitoren-Prozess) Gegenstand der Untersuchung und bildet somit zudem auch den Umfang.

Die Evaluierungskriterien wurden in Kapitel 1.5 zusammen mit den Anforderungen formuliert. Da es sich um die Erprobung eines neuen Testansatzes handelt, wurden zum besseren Verständnis in Kapitel 4 die textuellen Testfallspezifikationen als Basis der anschließenden Testfallmodellierung angefertigt. Auch sorgen die Spezifikationen beim Modellierer initial für ein besseres Verständnis bezogen auf den Hintergrund der Geschäftsprozesse. Bei der Testskriptprogrammierung wählt der Autor die Skripttechnologien: datengetrieben, schlüsselwortgetrieben oder aktionsgetrieben. So entsteht eine Mischung aus dem modellbasierten Testen, gepaart mit skriptbasierten Testautomatisierungstechnologien, welche zusammen das modellbasierte Testen von verteilten BIS ermöglichen. Als Testmanagementtool wird das HPQC vorgeschlagen und genutzt. Die programmierten Adaptierungen und Anpassungen im HPQC wurden in VBA entwickelt, so lassen sich diese Funktionsbausteine leicht auf andere Testmanagementsysteme übertragen.

5.1 Konnten die Ziele dieser Dissertation erreicht werden?

Die Anforderung an diese Dissertation war es, eine MBT-Methode für verteilte BIS auf der Basis von BPMN-Testmodellen zu entwickeln. So konnten die in Kapitel 1.4 formulierten Anforderungen (1–4) im Kapitel 2 „Stand der Technik“ herausgearbeitet werden. Sowohl UML als auch BPMN sind mächtige Modellierungsnotationen, wobei die Zielgruppe von BPMN-Testmodellen nicht technisch versierte Ingenieure oder Informatiker sind [92], sondern Fachbereiche mit wenig IT-technischen Kenntnissen [151], die aber umso mehr die Geschäftsprozesse im Unternehmen beherrschen. Die Anforderungen (5–9) konnten in den Kapiteln 3 und 4 erläutert und demonstriert werden. Gleichzeitig bilden die beiden Kapitel 3 und 4 den Kernteil dieser Dissertation. Es konnte gezeigt werden, wie sowohl das Modellierungswerkzeug IBA mit Hilfe von BPMN-Spezifikationserweiterungen und Konfigurationsdateien (Testprofilen) als auch die Testwerkzeuge MBTSuite, HPQC und QTP so integriert werden, dass Testfälle aus den Modellen ins HPQC übertragen und mit QTP und mit dem HPQC-Testscheduler ausgeführt werden können. Änderungen in Softwaresystemen, wie Prozesserweiterungen oder Prozessveränderungen, werden einfach im Testmodell nachgebildet. Die MBTSuite bietet die Möglichkeit, ein Testmodell nicht komplett zu generieren, sondern nur veränderte Knoten, Kanten oder Pfade zu betrachten und daraus Delta-Testfälle zu generieren. Ändert sich allerdings die Ablauflogik eines Geschäftsprozesses, so müssen notgedrungen auch die Testskripte angepasst oder es muss zumindest die Reihenfolge der einzelnen aufzurufenden Funktionsbausteine überarbeitet werden. In

erster Linie werden mit dieser Methode Regressionstests entwickelt, welche schrittweise erweitert und zu einer Testsuite zusammengefasst werden können. So kann nach jeder Releaseeinspielung diese Testsuite automatisiert ausgeführt werden, um sicherzustellen, dass durch neue Anpassungen und Erweiterungen in betroffenen Softwaresystemen keine negativen Auswirkungen auf das Funktionieren von Geschäftsprozessen erfolgen.

5.1.1 Entwicklungsphasen: von den Anforderungen zum ausführbaren Testskript

Die Entwicklungsphasen laufen immer nach demselben Schema ab, so muss eine Testanforderung existieren, aus der ein Testmodell angefertigt werden kann. Parallel muss der Business-Analyst immer Personen aus dem Fachbereich und IT-Experten bei der Modellierung einbinden.

- Testanforderungen werden direkt in ein Testmodell übertragen. Anders als ausnahmsweise in Kapitel 4 demonstriert, sollten die textuellen Testfallspezifikationen nicht mehr manuell angefertigt werden.
- Inhalt der Testprozessmodellierung sind auch die Spezifizierungen der einzelnen Prozesselemente sowie die für den Test vorgesehenen Testdaten. Sowohl die Dateneingaben als auch die Datenausgaben müssen spezifiziert werden, um einen Test der Schnittstellen zu ermöglichen. Die Spezifikation der Testdaten erfolgt sowohl in UML-Klassendiagrammen als auch im Excelformat, wobei Testautomatisierungswerkzeuge in der Lage sind, diese Testdaten im Excelformat auszulesen und zu verarbeiten. Testdaten können gleichzeitig mittels UML-Klassendiagrammen mit Hilfe von IBA spezifiziert und in das Excelformat oder aber auch in das Format XML konvertiert werden. So können die für die Testdaten notwendigen „Spaltennamen“ in Excel angelegt werden. Diese Spaltennamen sind die Attribute, welche mit Hilfe der Parameter jetzt gepflegt werden können.
- Testfälle werden nach ihrer Modellierung mittels IBA in die MBTSuite exportiert, dann erfolgt im nächsten Schritt die Selektierung der zu generierenden Testpfade. So können die Testpfade nach der Priorisierung traversiert und ausgeführt werden. Der Ansatz dieser Dissertation ist es, Testfälle automatisiert auszuführen. Daher werden die Testfälle so modelliert, dass diese im Grunde nur Testschritte repräsentieren, wobei die eigentliche Testlogik in die Funktionsbibliotheken verlagert wird. Ein großer Vorteil ist dabei, dass die Testmodelle kompakt und übersichtlich bleiben. Die Testschritte bezeichnen die Aktion, die durchzuführen ist, die Details sind in Funktionen gekapselt.
- Mit der im Rahmen dieser Dissertation vorgenommen Erweiterung um BPMN-Testelemente können Testschritte mit verschiedenen Stereotypen auch als Testschritte gekennzeichnet werden. Damit ist es möglich, sowohl Testschritte als auch Verifikationspunkte im Testmodell kenntlich zu machen, was es wiederum erlaubt, diese Elemente voneinander zu unterscheiden. Diese Erweiterungen sind in einem Testprofil zusammengefasst und können für beliebige Testprojekte

genutzt werden.

- Jeder generierte Testschritt repräsentiert im Hintergrund eine „gewisse“ Testlogik. Diese Testlogik ist in Funktionsbibliotheken implementiert und gekapselt. Bei der Generierung eines Testfalls aus einem Modell prüft die Adaptierung, ob eine korrespondierende Funktion in der Funktionsbibliothek existiert. Dabei spielen die festgelegten Namenskonventionen eine entscheidende Rolle. Sie dienen als Basis der Adaptierung und als Grundstein der hier vorgestellten Methode. So ist hier definiert, dass jedes zu testende System mit seinem Softwarekürzel beginnen muss. Damit können auch verschiedene Softwaresysteme mit Hilfe eines Testmodells modelliert und dargestellt werden. Bei der Generierung der Testfälle werden jeweils die benötigten Testskripte für den Testlauf aufgerufen.
- Funktionsbibliotheken beinhalten die Gesamtheit aller am Prozess (Geschäftsprozesse) beteiligten Testlogiken und können somit schrittweise um Softwaresysteme und Ablauflogik erweitert werden. So entsteht sukzessive ein Stamm an Funktionen, auf die immer wieder zurückgegriffen werden kann. Werden Prozesserweiterungen nötig, welche ein explizites Softwaresystem benötigen, so kann dieses beliebig hinzugefügt werden.
- Das HPQC bietet genügend Auswertungsmöglichkeiten, um ständig über den Stand der Testausführung eine Auskunft geben zu können.
- Die Konstellation von IBA und MBTSuite kann allerdings auch eine Fehlerquelle darstellen, und zwar genau dann, wenn die Modelle nicht eingelesen werden können. Das kann daran liegen, dass in BPMN Elemente hinzugefügt werden können, welche die MBTSuite nicht interpretieren kann. Mit Hilfe des entwickelten Testprofils ist dieses Problem zunächst gebannt.

5.1.2 Methodenzusammensetzung

Die im Rahmen dieser Dissertation vorgeschlagene Methode setzt sich aus verschiedenen Komponenten zusammen. Die Testwerkzeuge bilden dabei nur eine Facette. Zusätzlich müssen sowohl ein Testfallgenerator als auch ein BPMN-Modellierungstool angeschafft werden. Eine weitere Facette bietet die Erweiterung der BPMN-Spezifikation und des Testmanagementwerkzeuges um die Möglichkeit der Adaptierung. Inwiefern das hier erstellte Testprofil sich auch auf andere Modellierungswerkzeuge übertragen lässt, kann im Rahmen dieser Dissertation nicht untersucht werden. Ein Merkmal dieser Methode ist, dass sie aus einer Kombination von modell- und skriptbasierten Testautomatisierungstechnologien besteht. Dies macht die Methode robust gegenüber Änderungen an der zu testenden Software und sie kann jetzt sukzessive durch Erfahrungswerte aus der Praxis erweitert und verbessert werden. Zentraler Bestandteil der Methode sind die Testdaten, welche methodisch mit Hilfe von Äquivalenzklassen und Grenzwertanalysen ermittelt und spezifiziert worden sind.

5.2 Was sind die Vorteile der BPMN-Testmethode?

Der Grundgedanke des modellbasierten Testens ist es, ein Modell zu erstellen, welches das erwartete Verhalten des Systems beschreibt. Testfälle werden dabei als Pfade im Modell betrachtet, jeder Pfad repräsentiert einen Testfall. Bei der Ausführung der Tests wird das Softwaresystem mit den Eingabewerten aufgerufen, dabei werden die Ausgabewerte direkt mit den in den Exceltabellen hinterlegten, erwarteten Werten verglichen. Auf der Basis dieser Ergebnisse kann dann eine Aussage über die Konformität des Softwaresystems getroffen werden.

In der Domäne der verteilten BIS agieren komplexe Geschäftsprozesse miteinander, um eine Wertschöpfung für das Unternehmen zu erzielen. Das betriebswirtschaftliche Wissen hinter Geschäftsprozessen obliegt in den meisten Fällen den Fachbereichen. Diese sind neben gesetzlichen Vorgaben die Architekten dieser Softwaresysteme [142]. Sowohl Entwickler, die „nur“ die funktionalen Vorgaben in technischer Sprache umsetzen, als auch die Tester solcher komplexen Softwaresysteme hängen von den Erfahrungen und vom Wissen der Fachbereiche ab. Somit stellt sich die Frage: Wie können Fachbereiche bei der Erstellung von Testmodellen hinzugezogen werden? Die UML [25] ist ohne Frage eine mächtige Modellierungssprache, verfehlt allerdings ihre Zielgruppe, wenn es um Fachbereiche in Unternehmen mit wenig IT-technischem Hintergrund geht [151]. Die im Rahmen dieser Dissertation vorgestellte Modellierungsnotation BPMN versteht sich als Brücke zwischen Fachbereichen und IT [152], [151]. Zumal mehrere Fachbereiche ihre Geschäftsprozesse auf die Modellierung mit BPMN umstellen [152]. Da bietet es sich an, die Testfälle auch direkt in BPMN zu modellieren.

5.2.1 Einbindung von Fachbereich, IT-Experten und Testern

Die hier vorgeschlagene Methode bringt auch Seiteneffekte mit sich. Während der Anfertigung dieser Dissertation musste sowohl bei den Fachbereichen als auch bei Testern und IT-Experten ein Umdenken beim Thema Testen von Software stattfinden. Durch die frühzeitige Einbindung aller Parteien in der Projekt- und Entwicklungsphase können Synergien geschaffen werden, die sich später auf die Qualität des auszuliefernden Produktes positiv niederschlagen [133]. Die Idee: Fachbereiche lernen das Modellieren von BPMN-Geschäfts- und Testprozessen parallel, während ein Tester (Business-Analyst) zusammen mit einem Mitarbeiter aus dem jeweiligen Fachbereich die Testmodelle erstellt. Fehler, Widersprüche und Verbesserungsvorschläge, die sich so aus der Zusammenarbeit an einem Testmodell ergeben, lassen sich direkt im Testmodell identifizieren und beheben. Damit das Problem [145], [150] der unzureichenden Abstimmung zwischen IT und Fachbereich umgangen werden kann und die Fachbereiche nicht mit dem kompletten BPMN-Umfang hantieren müssen, wurde ein spezielles Testprofil für BPMN-Elemente durch den Autor dieser Dissertation erstellt. Dieses soll die Komplexität der Modellierung deutlich reduzieren.

5.2.2 Kürzere Modellierungszeiten

Durch die Einbeziehung aller am Projekt beteiligten Parteien konnte der Prozess der Modellierung von Testfällen um 50% beschleunigt werden. Durch den ständigen Review der Testmodelle durch alle Parteien kann parallel zur Modellierung eine Qualitätssicherung des Testmodells erreicht werden. Beim Vergleich der Zeiten zwischen der Modellierung eines Testfalls und der Erstellung eines textuell basierten Testfalls konnten im ersten Schritt keine messbaren Unterschiede identifiziert werden. Im zweiten Schritt und mit der Etablierung der Modellierung der Testfälle konnten diese Prozesse deutlich um die Hälfte beschleunigt werden. Allerdings konnten am Testmodell mehrere Testpfade identifiziert werden, welche zuvor in keiner Testspezifikation auftauchten, was einen Vergleich an dieser Stelle unmöglich macht. Es konnten am Fallbeispiel 105 zusätzliche Pfade identifiziert werden, wovon 15 Pfade (siehe Kapitel 5.2.6) zur Identifizierung von Fehlern führten, welche schon in Produktivumgebungen vorhanden sind.

Auch konnten zusätzliche Testfallspezifikationen hinzugefügt werden, wie die Tatsache, dass nach einem dreimaligen fehlgeschlagenen Anmeldeversuch bei einem Softwaresystem nicht nur automatisiert geprüft werden muss, ob dieser Benutzter tatsächlich gesperrt wird. Sondern zusätzlich werden Pfade generiert, welche das automatische Informieren eines Administrators bei mehr als drei fehlgeschlagenen Anmeldeversuchen testen.

5.2.3 Weniger VBA-Code in den Testskripten als bei einer klassischen Testautomation

Durch die Modellierung der Testschritte in Aktivitäten konnte zu jedem Testschritt eine Funktion hinterlegt werden, welche die Testlogik zu diesem Testschritt beinhaltete. Mit Hilfe der Testmodellierung können jetzt bspw. viele Verzweigungen identifiziert werden, welche auf dieselben Funktionsbausteine referenzieren. So konnten viele Redundanzen auch in bestehenden Funktionen eliminiert und dadurch viel Programmieraufwand gespart werden. Konkret konnten 384 Testfälle (siehe 5.5.1), welche für das Fallbeispiel vonnöten wären, in 130 Testskripten abgebildet werden. Jedes Testskript beinhaltet mehrere einzelne Testschritte, bildet aber in Summe immer einen Testfall. So konnten viele Redundanzen eliminiert und Synergien durch den modularen Aufbau der Testskripte genutzt werden. Durch die Namenskonventionen ist gleichzeitig der Programmiercode besser strukturiert. Durch den Aufbau der Funktionen aus einzelnen Funktionsbausteinen können Testfälle jetzt in einer beliebigen Testreihenfolge ausgeführt werden. Die Funktionsbausteine erlauben es gleichzeitig, sich untereinander aufzurufen, so kann ein Testfall direkt Daten aus einem CFM-System bspw. in die Kassensysteme laden. Somit ist dies der erste Testfall (lade Daten aus CFM) und der darauffolgende Testfall kann dann sein: Prüfe in Kassen übergebene Daten aus CFM.

5.2.4 Auslagerung der Testdaten im Excelformat und in UML-Klassendiagrammen

Das Auslagern von Testdaten kann auf der Basis von verschiedenen Lösungen erfolgen. Wird mit Datenbanken gearbeitet, ist es möglich, Testfälle mittels SQL[101] auf der Datenbank mit Werten zu versorgen. Es können bspw. die Extremwerte, die häufigsten Werte oder zufällige Werte ausgewählt werden. Im Rahmen dieser Dissertation wählte der Autor die Auslagerung von Testdaten in Exceltabellen, da diese ohne SQL-Befehle auskommen und durch Testskripte ausgelesen werden können. Zusätzlich bieten diese den Vorteil, dass Fachbereiche einfach Testdaten in Exceltabellen hinzufügen oder ändern können. Die Kompatibilität zwischen dem Testautomatisierungswerkzeug QTP und Excel erleichtert obendrein die Entscheidung, Testdaten im Excelformat abzulegen. Die generierten Testskripte werden während der Laufzeit mit Testdaten aus Exceltabellen versorgt. Das Speichern von Daten, die wiederverwendet werden sollen, ist vor allem für den Regressionstest wichtig. Bei der Spezifizierung der Testdaten durch Fachbereiche werden Ergänzungen und Erweiterungen vorgenommen, diese bleiben nach jedem Testlauf bestehen. So kann immer auf einen Bestand an Standardtestdaten zurückgegriffen werden [85].

5.2.5 Schrittweise Erhöhung des Testabdeckungsgrads

Bevor eine Testmethode, welche auf Modellen basiert, in einer Unternehmensorganisation eingeführt wird, ist es zunächst wichtig, zu prüfen, inwiefern Testprozesse, Teststufen und Testarten in der jeweiligen Unternehmensorganisation „gelebt“ werden und etabliert sind. Die Erhöhung eines Testabdeckungsgrads nach der Einführung von MBT als Methode kann nur nachgewiesen werden, wenn es zuvor schon Kennzahlen gibt, mit denen ein Vergleich möglich ist [104].

In der Evaluierungsphase konnte festgestellt werden, dass die Anzahl der aus den verschiedenen, hier am Fallbeispiel beteiligten Softwaresystemen generierten Testfälle deutlich höher ist als die zuvor manuell spezifizierten Testfälle. Durch die Kapselung der Testfälle in Testskripte konnten wiederum Redundanzen in den Testfällen vermieden und dadurch die Anzahl der letztendlich resultierenden Testskripte reduziert werden, bei gleichzeitiger Erhöhung des Testabdeckungsgrads. Die Anmelderoutine in einem beliebigen Softwaresystem wird dadurch getestet, dass ein Benutzer sich mit seinem Passwort und sonstigen Informationen wie Anmeldesprache, Client- und Systemauswahl anmelden kann. Mit Hilfe des entworfenen Testmodells sind jetzt insgesamt drei Testfälle hinzugekommen. Die Testfälle werden bewusst „provoziert“, um sicherzustellen, wie sich das Softwaresystem bei Anmeldefehlversuchen verhält. Das ist nur ein Beispiel.

Wie verhält sich das System bei einem dritten Fehlversuch? Nach dem dritten Fehlversuch muss ein Benutzer gesperrt werden. D. h., dass nach einem dritten Fehlversuch automatisch ein zusätzlicher Testfall generiert wird, welcher prüft, ob der Benutzer tatsächlich gesperrt ist.

Ein weiteres Beispiel ist das Verbuchen von Belegen in der Finanzbuchhaltung. Dies kann über verschiedene Transaktionen erfolgen. Im Testmodell werden alle möglichen Bu-

[101] Datenbank-Abfragesprache.

chungstransaktionen erfasst. All diese Transaktionen haben das Ziel des Buchens von Finanztransaktionen, mit der Unterscheidung, dass jede Transaktion einen anderen betriebswirtschaftlichen Hintergrund besitzt. Damit alle Buchungstransaktionen mindestens einmal getestet werden können, ist es nötig, sie zumindest in einem Testmodell sichtbar zu machen. Der Testfallgenerator generiert dann zu jeder dieser Varianten und Testpfade je nach definierten Bedingungen einen Testfall.

Ohne ein Testmodell wird keiner auf die Idee kommen, alle möglichen Transaktionswege, welche Buchungen vornehmen, zu testen. Erst wenn diese visuell erfasst werden, sieht ein Tester die Notwendigkeit, diesen Pfad auch einmal auszuführen. Das ist aber auch dem zeitlichen Druck geschuldet, welcher in Testphasen immer herrscht [10]. Es wird nur das getestet, was die jeweilige Person für den Praxisbetrieb (Tagesgeschäft) benötigt. Das modellbasierte Testen hingegen bringt hier eine deutliche Verbesserung, weil es den Testprozess insgesamt viel weiter vorne im Softwareentwicklungsprozess ansiedelt. Die Verbesserung lässt sich anhand der gefundenen zusätzlichen Fehler ableiten. Die Beschleunigung des Testprozesses lässt sich auch nach mehreren Durchläufen auf 50% beziffern.

Auch bringt die schrittweise Automatisierung jeder Transaktion Vorteile. Sie beschleunigt nicht nur den Testprozess, sondern bietet gleichzeitig die Möglichkeit, automatisierte Funktionsbausteine immer wieder in verschiedensten betriebswirtschaftlichen Konstellationen auszuführen.

5.2.6 Frühzeitige Fehleridentifizierung und -behebung

Mit dem Ansatz konnten Fehler schon in der Modellierungsphase eines Testfalls identifiziert werden. Beim Design der Testschritte und bei der Modellierung der Abfolge konnten Mitarbeiter von Fachbereichen Hinweise erhalten, wie bspw. die automatische Sperrung eines Benutzers nach drei Anmeldefehlversuchen. Ein kritischer Fehler konnte bei der Spezifizierung der Testdaten entdeckt werden. So konnte durch den Einsatz von Äquivalenzklassen und Grenzwerten festgestellt werden, dass die Regelung zwischen Komma und Punkt bei Summenbeträgen nicht klar und ausreichend in der Anforderungsspezifikation spezifiziert war.

Softwaresysteme	gefundene Fehler in der Modellierung	gefundene Fehler bei Anwendung Methode
Kundenmanagementsystem	42	21
Kreditsystem	77	19
Kassensystem	57	30
Warenwirtschaftssystem	20	4
SAP System	27	12
Bankensystem (integriert)	2	1
Summe	225	87

Tabelle 2: Aufstellung der gefundenen Fehler

Die obige Tabelle gibt eine Zusammenfassung der während der Evaluierungs- und Erprobungsphase beobachteten Fehler. Die meisten Fehler und Abweichungen konnten beim

Kreditsystem identifiziert werden, da dieses Softwaresystem, wie schon mehrmals erwähnt, sehr speziell ist. Die Finanzrichtlinien eines Unternehmens müssen sich vollständig in der Softwarelösung abbilden, inklusive aller Genehmigungsstufen je nach Kredithöhe und Kreditgrenze. Bei der Testfallmodellierung konnten systemübergreifend 225 Fehler sowohl in der Testfallspezifikation als auch in der Anforderungsspezifikation gefunden werden, die ohne den Ansatz der Modellierung unmittelbar in die Testfälle eingeflossen wären. Bei der Anwendung der Testmethode konnten 87 Fehler gefunden werden, die mit der klassischen Testausführung nicht identifiziert worden wären. Bei den 87 Fehlern konnte ermittelt werden, dass 22 Fehler sich schon in Produktivumgebungen befanden und diese durch die klassischen Ansätze beim Testen nicht identifiziert werden konnten. Diese Lücke konnte jetzt geschlossen werden, was nicht bedeutet, dass die Produktivsysteme nun fehlerfrei sind.

Von den 22 Fehlern waren 3 kritische Fehler, die bei den Fachbereichen bisher zu einem Mehraufwand beim täglichen Arbeiten führten. Dem Fachbereich war an dieser Stelle auch nicht klar, dass es sich um einen klassischen Softwarefehler handelte. So konnten die Fehlerwirkungen mit einem zusätzlichen Arbeitsaufwand kompensiert werden. Durch die neue Testmethode konnte hier eine neue Anforderung formuliert werden, welche genau eine Lösung für diesen Fehler bietet und diesen gleichzeitig behebt.

5.3 Anwendungen in der Praxis – Methodenevaluierung mit Hilfe von B2B-Prozessen im Handel

Die Validierung eines methodischen Ansatzes hinsichtlich eines möglichen Vorteils in der Praxis des Softwaretestens muss letztlich auch anhand einer ökonomischen Betrachtung geführt werden. Dazu wäre der Nachweis zu führen, dass der zusätzliche Nutzen der Methode die nötigen Mehraufwände im Vergleich zu alternativen Ansätzen übersteigt. Eine derartige Evaluierung ist für Testmethoden im Allgemeinen kaum durchführbar. Dies hat seine Ursache zum einen darin, dass in einer Kosten-Nutzen-Analyse vor allem auch die Folgekosten nicht entdeckter Fehler bewertet werden müssen. Die Frage der Wirtschaftlichkeit ist daher heute auch für den modellbasierten Test im Allgemeinen noch nicht abschließend beantwortet [30].

Das Fehlen einer aus dem wirtschaftlichen Nutzen abgeleiteten Kennzahl für Testsuiten ist ein Grund dafür, andererseits erfordert eine empirische Absicherung das Vorliegen umfangreicher Experimente. Repräsentative Fallbeispiele, welche hierzu ausreichend Daten (wie bspw. Vergleichstestfälle, Angaben zu nicht entdeckten Fehlern, Erhebungen zu Fehlerfolgekosten) umfassen, sind kaum verfügbar und in der Durchführung stellen sich prinzipielle Schwierigkeiten ein, insbesondere hinsichtlich der Gewinnung allgemeingültiger Aussagen.

Es ist auch nicht Anspruch der vorgeschlagenen Methode, dass durch ihre Anwendung Testfälle ermittelt werden können, welche eine höhere Fehlerentdeckung garantieren. Das betrachtete Gütekriterium ist dagegen die Eignung der Testfälle und Testsuiten für den Test der spezifizierten Einzelanforderungen. „Eignung“ ist hier vor allem derart zu verstehen, dass die Menge der Testfälle geeignet ist, um den Test der einzelnen Anforderungen gegenüber den „Kunden dieser Systeme“ (Fachbereiche) nachweisen zu können, und ins-

besondere, dass das vorgeschlagene Verfahren hierzu vorteilhaft im Vergleich zu anderen Testmethoden ist.

Die hier vorgeschlagene Testmethode ist gegenüber der klassischen Methode vorteilhafter, weil sie nicht nur den Testprozess beschleunigt, wie oben schon erläutert, sondern weil sie auch Vorteile bietet, wie die automatisierte Generierung von Testfällen und die gleichzeitige Steuerung der tatsächlich gewünschten Testabdeckung. Auch bietet sie die Möglichkeit, vorher noch unbekannte Fehler als Fehler zu identifizieren, da die visuelle Darstellung eines Testfalls als Grundlage von Diskussionen zwischen Fachbereichen, IT und Testern genutzt werden kann.

5.3.1 Wie wurde geprüft – wer hat die Methode ausprobiert?

Die hier dargestellte Methode kann in beliebigen B2B-Unternehmen unter Praxisbedingungen evaluiert werden. Die Methode wird im Rahmen der regulären Releasetests in der Regressionstestphase eingesetzt. Dabei sind die modellierten und generierten Testskripte über den Testscheduler vom Autor dieser Dissertation gestartet und durch den Testautomaten ausgeführt worden.

Testfälle waren schon immer Bestandteil des Qualitätsmanagements und eines funktionierenden Testprozesses innerhalb der Unternehmensorganisation. Die im Rahmen dieser Dissertation ausgearbeitete Methode bringt allerdings eine Neuausrichtung der IT-Qualitätssicherung mit sich. So sind bei dieser Methode Unternehmensorganisationen gezwungen, sowohl Fachbereiche, Tester als auch IT-Experten in einer frühen Entwicklungsphase zusammenzubringen. Durch die Erstellung von Testmodellen und durch die Adaptierung von wiederverwendbaren Testskripten sind von Fachbereichen gute Ergebnisse sowohl bei der Testfallspezifizierung als auch bei der späteren Testabnahme erzielt worden. Angereichert werden die generierten Testskripte mit extern gelagerten Testdaten. Diese Daten wurden zusammen mit Mitarbeitern aus Fachbereichen applikationsübergreifend ergänzt und mit Hilfe von Äquivalenzklassen spezifiziert.

Diese Methode wurde im Rahmen von Regressionstests erprobt und es sind schrittweise Verbesserungen und Erweiterungen an den Funktionsbibliotheken vorgenommen worden. Somit sind stabile und robuste Testmodelle und Testskripte entstanden. Auch die Testmodelle können nach mehrmaligem Ausprobieren in der Praxis deutlich schneller (50%) angefertigt werden. Die hier eingesparte Zeit kann in andere Testaktivitäten investiert werden. Auch so eine Maßnahme trägt zur Steigerung des Testabdeckungsgrades bei. Gleichzeitig unterstehen diese Modelle ständig einem „Review“ sowohl durch den Fachbereich als auch durch die an den Tests beteiligten Personen.

5.3.2 Welchen Umfang hatte die Evaluierung?

Bei der Evaluierung muss darauf geachtet werden, dass eine produktnahe Umgebung mit allen am Prozess beteiligten Softwaresystemen gegeben ist. Nur so ist sichergestellt, dass ein Test unter realen Bedingungen stattfinden kann und dadurch eine Aussagekraft über die Qualität der Produktivsysteme getroffen wird. Die Infrastruktur für die Evaluierung war gegeben, so konnte die Erprobung unter Praxisbedingungen erfolgen. Ausgenommen ist

die ständige Onlineverbindung zu externen Wirtschaftsauskunfteien. Hier sind Standardwerte in die Testobjekte eingesetzt worden. Die Evaluierung erfolgte anhand der folgenden Abbildung und der Prozessstruktur, welche die IT-gestützte Zahlungsabwicklung darstellt.

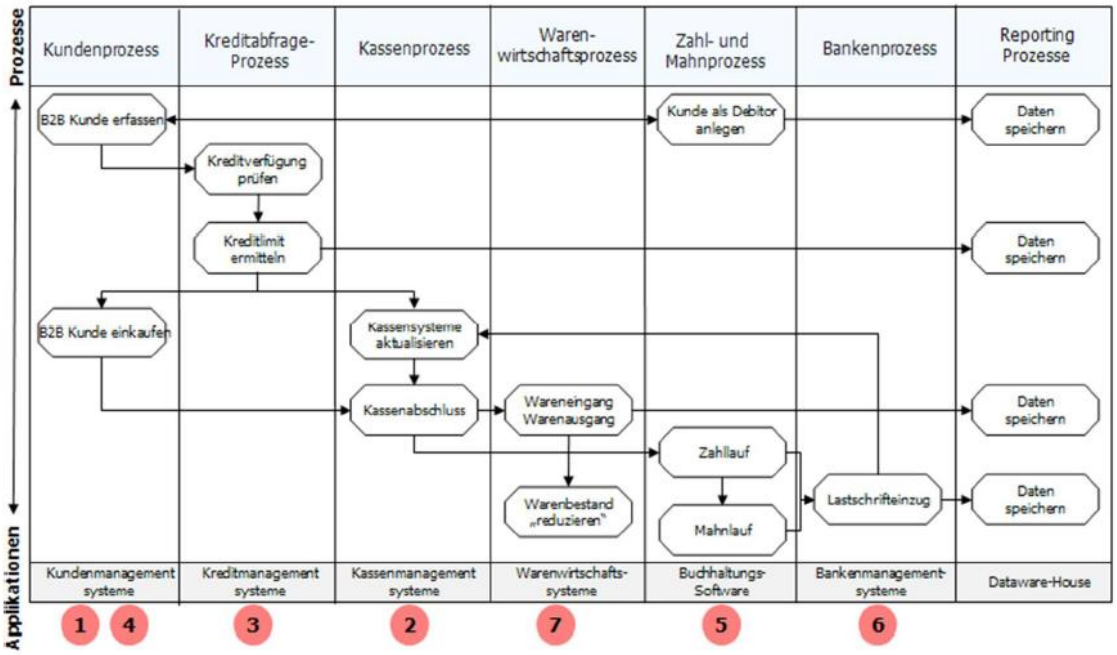


Abbildung 76: Prozesse im IT-gestützten Zahlungsabwicklungsprozess

Untermauert wird der Prozess gleichzeitig mit der in Kapitel 4 erläuterten Testfallbeschreibung. Bei der Modellierung der ersten Testfälle konnten Fachbereiche direkt Verbesserungsvorschläge einbringen, wie das variantenreiche Testen von Buchungstransaktionen, welches die Testabdeckung erheblich erhöht. Diese Hinweise sind sofort im Testmodell eingearbeitet worden.

Der Ausbildungsstand der Fachbereiche bzgl. der Modellierung mit BPMN liegt jeweils in den Grundzügen, sprich, es sind Schulungen notwendig. Gemeinsam mit Business-Analysten konnten allerdings die ersten am Prozess beteiligten Testmodelle reibungslos angefertigt werden. Beginnend mit dem „Anlegen" von Kunden in einem CFM-System wurden dann die benötigten Schnittstellen mit den angelegten Kunden und Kundennummern versorgt. Auf der Basis des implementierten Funktionsbausteins werden jetzt verschiedene für den Test benötigte Kunden angelegt. Zusätzlich wird durch die Testdaten gesteuert, welche Variante beim Testen genutzt werden soll.

5.3.3 Methoden von Handhabung und Nutzung

Eine Schulung der Methodennutzer ist unabdingbar. Fachbereiche und Tester, die diese Methode nutzen und anwenden, müssen eine Schulung sowohl in der Modellierung mit BPMN als auch im Umgang mit dem Testfallgenerator erhalten. Sowohl das Modellieren als auch das Generieren der Tests bilden nur einen Teil. Der zweite Teil besteht darin, fertig automatisierte Testskripte aus den generierten Testfällen zu gewinnen. Dies geschieht durch Adaptierung. Die Implementierung dieser Funktionsbausteine, welche die Adaptierungen ermöglichen, erfolgt durch einen Business-Analysten (Tester).

Weil beide Gruppen miteinander arbeiten müssen, wachsen sowohl der Business-Analyst als auch die Mitarbeiter aus den Fachbereichen stetig an ihren Aufgaben. Die Modellierung der Testfälle in BPMN konnte nach einer Einarbeitungszeit von ca. drei Arbeitstagen erste Erfolge vorweisen. Da die BPMN-Elemente nur auf das Nötigste reduziert werden, ist das Anlernen relativ leicht. Bei der Testfallgenerierung mit der MBTSuite ist es nötig, fünf bis sieben Arbeitstage zu investieren. Damit bekommen die jeweiligen Personen die grundlegendsten Funktionen dieses Werkzeugs erläutert und können selbstständig die ersten Testmodelle entwerfen.

Das Arbeiten mit QTP erfordert Expertenwissen und Programmiererfahrungen. Hier ist ein gut ausgebildeter Softwaretester mit Testautomatisierungsexpertise erforderlich. Dieser kann wiederum zusammen mit den Fachbereichen und mit dem Business-Analysten die generierten Testfälle an Testskripte adaptieren und somit einen Pool an Funktionen erstellen.

Es empfiehlt sich eine Arbeitsteilung zwischen Business-Analyst, Fachbereich und einem IT-Experten (Testautomatisierungsexperte). Diese Vorgehensweise ist in der Evaluierung erprobt und brachte sehr gute Erfolge, zumal alle Beteiligten ihre Expertise direkt einbringen können. Beim Testen der Testskripte fallen wiederum Verbesserungen an, welche sich in den Testmodellen wiederfinden müssen.

Generell muss bei der Einführung der hier vorgeschlagenen Methode darauf geachtet werden, dass schon die Einführung und die Implementierung dieser Methode als Softwareprojekt betrachtet und mit einem Projektplan versehen werden müssen. Sonst besteht die Gefahr, dass die Methode schon an administrativen Verantwortlichkeiten scheitert. Ein Projekt muss dementsprechend aufgesetzt werden. Das hat den Vorteil, dass diese Methode auf sämtliche Unternehmensorganisationen erweitert werden kann und sowohl Fachbereiche, Business-Analysten und Tester bei der Einführung mitwirken können. Gleichzeitig werden die beteiligten Gruppen mit Schulungen und Seminaren zum Themenschwerpunkt versorgt.

Die Methode liegt als eigenständiges Projekt im HPQC und kann jederzeit von Experten genutzt werden, um betriebswirtschaftliche Geschäftsprozesse zu testen.

5.4 Ergebnisse: Ausführung der Regressionstestfälle für den IT-gestützten Zahlungsabwicklungsprozess

Zu jedem Testfall existiert im HPQC eine Testanforderung. Diese Testanforderung leitet sich aus dem Testmodell ab. Die beiden Aktivitäten „Testfallanforderung erfassen“ und „Testmodellierung“ dienen der Spezifikation und der Erstellung der Testfälle. Auf der Basis der Testmodelle sind die einzeln auszuführenden Testschritte zur Ausführung und zur Erreichung des geplanten Testziels in Aktivitäten modelliert. Jeder Teilprozess je Softwaresystem wurde zunächst in sich modelliert. Diese Teilmodelle bilden wiederum Testfälle und Testskripte ab, welche dann in der Evaluierung zum Einsatz kommen.

5.4.1 CFM-Kundensystem: Testmodelle und Testskripte in der Praxisanwendung

Beim CFM-Kundensystem werden drei Teilmodelle angefertigt:

Testmodell	Testfälle
CFM_Anmelderoutine	6 Testfälle

Von den sechs Testfällen, welche bei der Anmelderoutine generiert werden, entfallen drei auf die Verifikationsprüfung. Drei zusätzliche Testfälle können hinzukommen, wenn die Anmeldung, inklusive Anmeldefehlversuchen, simuliert werden soll. Hier erfolgt automatisch ein Test, der demonstrieren soll, dass das System tatsächlich automatisiert eine Information an einen Administrator sendet und ihn über den gesperrten User informiert. Dieser Test war bislang in keiner Spezifikation vorhanden, Dank der Testmodellierung kann jetzt auch dieser Prozess getestet werden.

Wichtig ist an dieser Stelle, dass sich hinter jedem Testfall (Testschritt im Modell) definierte Testablauflogiken verbergen. Die Ablauflogiken sind nicht Bestandteil des Modells und auch nicht der manuellen Testfallgenerierung. Ablauflogiken sind jeweils im Testskript implementiert und übernehmen zusammen mit im Testmodell angehefteten Testdaten die Ausführung der Tests. Die hier einmal implementierte Funktion für die Anmelderoutine (sowohl Aufruf als auch Durchführung der Anmeldung am System) kann jetzt beliebig oft für die Testaktivitäten genutzt werden.

Testmodell	Testfälle
CFM_Kunden_anlegen	16 Testfälle

Beim Prozess „Kunden anlegen“ umfasst der Standardtest 16 Testfälle pro einzeln anzulegendem Kunden im System. Hier zeigt sich schon der Vorteil der vorgeschlagenen Methode. Durch die Auslagerung der Testdaten können jetzt beliebige Kundendaten zusätzlich als Testdaten erfasst werden. Das zu implementierende Testskript ist dann in der Lage, alle Varianten und die dahinter liegenden Testdaten auszuführen. Der im Testmodell CFM_Kunden_anlegen modellierte Prozess umfasst auch das „Sperren“ und das „Entsperren“ von Kunden.

Testmodell	Testfälle
CFM_Kundendaten_übergeben	10 Testfälle

Beim Testmodell der Kundendatenübergabe an angebundene Systeme muss darauf geachtet werden, dass die Kundendaten tatsächlich auch auf den Zielsystemen ankommen. Dazu können wiederum die im Rahmen dieser Dissertation angefertigten Prüfbausteine für Testdaten genutzt werden. Diese sorgen schon vor Testbeginn dafür, dass die Daten validiert werden. Ein anderer Weg, um dieses zu überprüfen, ist einfach zu versuchen, die soeben erfassten Kundendaten im Zielsystem aufzurufen und zu verarbeiten.

Zusammenfassend lässt sich an dieser Stelle festhalten, dass eine Prozessüberdeckung mit Hilfe von 32 Testfällen im CFM erreicht werden kann. Die Prozessüberdeckung berücksichtigt sowohl die Anlage eines Kunden als auch die gleichzeitige Weiterleitung an die angebundenen Schnittstellensysteme. Die Anzahl der hier ermittelten Testfälle entspricht

jeweils dem Testlauf einer Variante. Nimmt ein Business-Analyst eine weitere Variante hinzu, so verdoppelt sich die Anzahl der Testfälle usw. Weiter lässt sich festhalten, dass eine Anzahl von 32 Testfällen im Rahmen des IT-gestützten Zahlungsabwicklungsprozesses ausreichend für einen 100%igen Testabdeckungsgrad ist. Die hier aufgeführten 100% beziehen sich ausschließlich auf die in der Testspezifikation geforderten Anforderungen, welche ein Testfall besitzen muss. Werden zusätzlich Freiheitsgrade spezifiziert und generiert, so kann sich die Anzahl der generierten Testfälle auf 93 erhöhen. Die Differenz kann ausgeführt werden; dies ist für den Prozess nicht notwendig, für einen vollständigen Test aber sicher zu empfehlen.

Die Testfälle sind zusammen mit den Fachbereichen für den Anwendungstest spezifiziert worden und bieten dadurch die bestmögliche Abdeckung im Kundensystem.

Die Anzahl der Testskripte ergibt sich aus den tatsächlich implementierten Funktionen in der Funktionsbibliothek. Auch konnte beobachtet werden, dass einige Funktionen und Funktionsinhalte sich überschneiden und dadurch einfach durch „Copy & Paste“ oder durch eine Referenz auf die Funktion übernommen werden konnten.

5.4.2 Kreditsystem: Testmodelle und Testskripte in der Praxisanwendung

Beim Kreditsystem ist das Testen etwas komplizierter. Für die im Rahmen dieser Dissertation notwendigen Daten reicht es, drei Testmodelle anzufertigen. In der Praxis agiert das Kreditsystem als Blackbox.

Testmodell	Testfälle
CM_Anmelderoutine	6 Testfälle

Der Ablauf dieses Testfalls ist analog zum CFM, bekommt aber einen anderen Stellenwert durch die Datensensibilität, welche solche Systeme besitzen.

Testmodell	Testfälle
CM_Kundendaten_laden	13 Testfälle

Das Testmodell des Ladens eines Kunden aus dem Kreditsystem besteht aus mehreren Teilschritten. In erster Linie wird getestet, ob die vom CFM übergebenen Daten via Schnittstelle Kreditsystem->CFM funktionieren. Ist das nicht der Fall, scheitert der Testfall schon an dieser Stelle und wird durch einen Fehler beendet. Durch die automatisierte Verifikationsprüfung kann untersucht werden, ob die korrekten Kundendaten tatsächlich geladen sind. Auch hier verbergen sich hinter jedem Testfall mehrere Testschritte, implementiert als Testablauflogik in der Funktionsbibliothek. Das Ergebnis dieses Testschrittes ist es, dass ein gewünschter Kunde im Kreditsystem verfügbar ist und dass die dort vom CFM übertragenen Daten zusätzlich mit soziologischen Daten erweitert werden, welche für eine Bonitätsprüfung erforderlich sind.

Testmodell	Testfälle
CM_Kreditlimit_berechnen	6 Testfälle

Das Teilmodell, welches den Prozess der Berechnung illustriert, ist abhängig von Fremdsystemen. So müssen Onlineverbindungen hergestellt werden, um Daten von zentralen Wirtschaftsauskunfteien abzufragen. Da dies nicht so einfach möglich ist, verzichtet der Autor dieser Dissertation auf die Erläuterung dieses Teilprozesses. Es wird angenommen, dass eine Blackbox genutzt wird, die auf der Basis von eingegebenen Kundendaten ein Kreditlimit zurückgibt.

Testmodell	Testfälle
CM_Kreditlimit_übertragen	12 Testfälle

Bei der Übertragung des ermittelten Kreditlimits an die angebundenen Systeme kann ein Standardvorgehen ermittelt werden. So werden immer die gleichen Systeme mit Daten bedient. Jetzt erfolgt direkt in den Testskripten eine Überprüfung der übergebenen Daten. Es wird konkret überprüft, welche Parameter und Attribute übergeben werden. Kapitel 4 zeigte die implementierten Funktionen, welche in der Lage sind, solche Prüfungen automatisiert und zur Laufzeit durchzuführen. Auch bei eingehenden Daten erfolgt vorher als Vorbedingung eine Überprüfung der Daten auf Format und Konsistenz. Dadurch kann erreicht werden, dass Fehler, welche auf fehlende oder korrupte Testdaten zurückzuführen sind, komplett verhindert werden können.

Analog zum Kundensystem sei erwähnt, dass die hier aufgeführten Testfälle nur für den Prozess des IT-gestützten Zahlungsabwicklungsprozesses vollständig sind. So konnte zusammen mit den Fachbereichen ermittelt werden, dass für die Übertragung der Daten aus dem Kreditsystem jeweils die generierten 12 Testfälle ausreichend sind. Hier bedeutet die Anzahl 12, dass pro Testlauf 12 Testfälle ausgeführt werden müssen.

5.4.3 Kassensystem: Testmodelle und Testskripte in der Praxisanwendung

Testmodell	Testfälle
Cash_System_login	6 Testfälle

Die automatisierte Anmelderoutine erfordert insgesamt 6 Testfälle.

Testmodell	Testfälle
Cash_Kundendaten_aufrufen	1 Testfall

Kundendaten laden mit Hilfe der Kundennummer des Kunden.

Testmodell	Testfälle
Cash_Scanvorgang_beginnen	20

Beim Scanvorgang wird auf die in den Testdaten hinterlegten Artikel zurückgegriffen und diese werden automatisiert mit Hilfe von QTP und mit den Artikelnummern in das Kassensystem eingepflegt. Dabei werden Prüfungen notwendig, nicht nur welche Artikel etc. erfasst worden sind, sondern auch Prüfungen hinsichtlich Rabatt, Menge, Brutto- und Nettopreis usw.

Testmodell	Testfälle
Cash_Kassenabschluss	12 Testfälle

Der Kassenabschluss ist ein ziemlich kompliziertes Verfahren, welches von den Kassen angestoßen wird. Die zuvor erstellten Funktionsbibliotheken enthalten die hier durchzuführenden einzelnen Testschritte.

5.4.4 Warenwirtschaftssystem: Testmodelle und Testskripte in der Praxisanwendung

Das Warenwirtschaftssystem muss die gemeldeten Daten aus den Kassensystemen verarbeiten und die Anzahl der Artikel und Waren nach dem Verkauf reduzieren.

Testmodell	Testfälle
WWS_System_Artikel_aktualisieren	15 Testfälle

5.4.5 SAP-FI: Zahlprozess

Der Zahlprozess ist einer der entscheidenden Prozesse im IT-gestützten Zahlungsabwicklungsprozesses. Durch den Zahlprozess werden Prozesse wie das Lastschriftverfahren oder der Mahnprozess angestoßen.

Testmodell	Testfälle
SAP_Anmelderoutine	6 Testfälle
Testmodell	Testfälle
SAP_System_Kassendatei_einlesen	30 Testfälle

Die Anzahl der Testfälle errechnet sich dadurch, dass SAP-Systeme nicht nur einen Kassenserver abfragen, um an die jeweilige Kassendatei heranzukommen, sondern alle angebundenen Kassenclients. Die dazu zu nutzende Testlogik ist immer gleich, aber die Testdaten variieren. Beim Anlegen der Zahlungsvorschläge kommt es immer darauf an, wie hoch die Anzahl der Buchungskreise ist, für welche ein Zahlungsvorschlag angelegt werden muss. Ein Zahlungsvorschlag beinhaltet 6 Testfälle, multipliziert mit den hier durchzuführenden Buchungskreisen (5) summieren sich die Testfälle schon auf 30.

Testmodell	Testfälle
SAP_Zahlungsvorschlag_anlegen	30 Testfälle

Neben dem Zahlprozess kommt der Lastschriftprozess zum Einsatz, welcher direkt vom Zahlprozess ausgelöst wird. Diesen Prozess zu testen, erfordert pro Buchungskreis 22 Testfälle, insgesamt 110 Testfälle.

Testmodell	Testfälle
SAP_Lastschriftprozess_starten	110

5.4.6 SAP-FI: Mahnprozess

Der Mahnprozess wird genauso wie der Lastschriftprozess vom Zahlprozess ausgelöst, und zwar genau dann, wenn Lastschriftprozesse nicht verarbeitet werden konnten. Die im Mahnprozess durchzuführenden Schritte ähneln denen im Zahlprozess, mit dem Unterschied, dass hier ein Mahnlauf [33] agiert.

Testmodell	Testfälle
SAP_Zahlliste_einlesen	5
Testmodell	Testfälle
SAP_Mahnprozess_Stufe 1_starten	10
Testmodell	Testfälle
SAP_Mahnprozess_Stufe 3_starten	12
Testmodell	Testfälle
CFM_Kundendaten_aktualisieren	12
Testmodell	Testfälle
SAP_Debitoren_sperren	12

5.4.7 SAP-Berechtigungsprüfung

Die Berechtigungsprüfung gilt im Rahmen dieser Dissertation als Nebeneffekt. Während der Ausarbeitung ließen sich viele Fehler in der Produktion auf fehlende oder unzureichende Berechtigungen zurückführen. Das SAP-System enthält ein mächtiges Berechtigungskonzept, welches nicht nur konfigurierte Unternehmensrichtlinien beinhaltet, sondern auch gesetzliche Vorgaben. So konnte ein Testskript entwickelt werden, welches das komplette Berechtigungswesen im SAP-System korrespondierend zu jedem erfassten Benutzer im System testet. Das Testmodell diente hier als Denkanstoß, um überhaupt Berechtigungsprüfungen vorzunehmen.

Testmodell	Testfälle
SAP_Berechtigungsprüfungen	12

Mit der Berechtigungsprüfung werden zwar „nur“ 12 Testfälle generiert und diese werden sogar nur auf einem Testskript abgebildet, aber dafür testet dieses Konstrukt ca. 44.000 Programme in 22 Stunden. Dies ist genau die Anzahl an Programmen, die in Summe von allen verfügbaren Benutzern in SAP-FI weltweit genutzt werden. Dieses Testskript wird zurzeit von der SAP analysiert, um es später ins SAP-Standardpaket zu integrieren. Der Vorteil dieses Testskripts ist, dass es auf alle SAP-Module übertragen werden kann, auch auf Nicht-SAP-FI-Systeme usw. Die Anzahl der hier generierten Testfälle kann schwanken und es können Testpfade sowohl hinzukommen als auch wegfallen. Die Anzahl von 384 Testfällen splittet sich auf 130 Testskripte auf. Ein Testskript beinhaltet zumeist mehrere zusammengefasste Testfälle.

5.4.8 Fehleranalyse

Sowohl während der Modellierung als auch während der Ausführung der Testfälle konnten fachliche Fehler identifiziert werden (5.2.6). Ob diese Fehler auch in einem manuellen Test ohne Modelle und Testautomation aufgefallen wären, kann nicht abschließend beurteilt werden. In der Evaluierung der Methode sind diese Fehler beim klassischen Testen nicht aufgefallen, sondern erst nach der Anwendung der hier vorgestellten Methode. Eine Abschätzung, wie viel mehr Fehler durch die Methode gefunden werden, kann erst nach mehrmaligem Einsetzen der Methode erfolgen. Auch müssen noch viel mehr Testprozesse mit dieser Methode modelliert und in die Automation eingebunden werden, um eine nachhaltige Analyse durchführen zu können. Das Kapitel 5.2.6 erläuterte die Fehlerentdeckung mit Hilfe dieser Methode. Nachweislich konnten 87 zusätzliche Fehler während der Testausführung identifiziert werden, welche unmittelbar Einfluss auf Produktivumgebungen haben. 22 dieser Fehler konnten zwar identifiziert werden, gleichzeitig musste aber auch festgestellt werden, dass diese Fehler schon in der Produktivumgebung vorhanden waren. Von diesen Fehlern mussten drei durch die damit verbundene Geschäftskritikalität direkt behoben werden.

5.5 Kosten-Nutzen-Analyse

Das Testen ist ein integraler Bestandteil von Softwareentwicklungsprojekten und leistet einen erheblichen Beitrag zur Sicherstellung der Qualität in Softwaresystemen. Der Anteil der Kosten der Qualitätssicherung an den Gesamtkosten großer Entwicklungsprojekte, insbesondere im Bereich der verteilten betrieblichen Informationssysteme, und auch die Kosten für das nachträgliche Beheben von Fehlern haben in den vergangenen Jahrzehnten stetig zugenommen [108]. Verfahren zur Testautomatisierung und damit auch die hier vorgestellten Verfahren für das modellbasierte Testen mit BPMN können schrittweise diesem Missverhältnis entgegentreten.

Wenn ein Unternehmen in die Verbesserung seiner Softwaretestprozesse durch eine neue Methode oder Testvorgehensweise investiert, möchte es wissen, ob sich diese Investition auch auszahlt. Eine hierfür wichtige Metrik ist das ROI (Return on Investment). Im Bereich des Testens können als ROI die eingesparte Zeit, das gesparte Geld und die höhere Qualität der Tests bezeichnet werden. Positive Nebeneffekte können zufriedene Mitarbeiter sein, die sich aufgrund von Veränderungen im Testprozess weiterentwickeln und ihre Ausbildung erweitern. Bspw. illustrieren die Autoren in [31], für welche Problemstellungen sich eine MBT-Testmethode eignet und welche Problemstellungen sich besser durch manuelles Testen oder durch die klassische Testautomatisierung lösen lassen. Die in dieser Dissertation vorgeschlagene Methode ist eine Mischung aus MBT und skriptbasierter Testautomatisierung, welche für die hier betrachtete Problematik die bestmögliche Lösung darstellt. Für die hier zu betrachtende Kosten-Nutzen-Analyse werden die vom Modell generierten Testfälle inklusive Adaptierung und Programmierung der Testskripte als Vergleichsgegenstand genommen. Dem gegenüberzustellen sind die Tests, welche bisher nur mit manuellem Aufwand ermittelt und ausgeführt werden konnten. Zusätzlich müssen die anzuschaffenden Test- und Modellierungswerkzeuge hinzugerechnet werden.

5.5.1 Kosten für Lizenz und Wartung

Die Entwicklung der hier vorgestellten Methode entstand im Rahmen von Testaktivitäten und Untersuchungen an komplexen und verteilten betriebswirtschaftlichen Anwendungssystemen. Die Entwicklung der hier vorgestellten Testmodelle und der Testskripte musste so geplant werden, dass neben der Entwicklung und Erprobung dieser Methode der Tagesbetrieb während der Testphasen fortgeführt werden konnte.

Was nicht unterschätzt werden darf, ist der hohe Aufwand, welcher investiert werden muss, um bspw. alle am Prozess beteiligten Softwaresysteme nicht nur so zu konfigurieren, dass ein einheitliches Testen möglich ist, sondern dass während der deskriptiven Programmierung sämtliche Komponenten einer Software in einer Funktionsbibliothek aufgefangen werden. Jeder Button im Kundensystem und im SAP-System usw. ist in einer Funktionsbibliothek als Komponente hinterlegt und kann beim Automatisieren dieser Prozesse immer herangezogen werden. Dieses Vorgehen eignet sich auch für Nicht-SAP-Systeme, in erster Linie, um alle für den Prozess wichtigen Komponenten in einer Funktionsbibliothek zusammenzufassen. Sprich, der Eigenanteil, um diese Methode zu verwirklichen, ist enorm hoch und schwer in einer Zeit auszudrücken.

Neben der menschlichen Komponente kommen die softwaretechnischen Aspekte hinzu. So müssen zumindest ein Modellierungswerkzeug (in dieser Dissertation wird IBA vorgeschlagen), ein Testfallgenerator, ein Testmanagementtool und ein Testautomatisierungswerkzeug angeschafft werden. Die Stundensätze ermitteln sich aus dem nationalen Standard für zertifizierte Testmanager, Tester, Softwareberater und Mitarbeiter aus Fachbereichen. Durch eine Mischkalkulation[102] konnte ein Durchschnittswert je Softwaresystem ermittelt werden. Ein Tester aus dem SAP-Bereich wird anders vergütet als bspw. Berater aus anderen Softwaresystemen.

Die folgende Tabelle gibt eine Aufstellung der Initialkosten, zusätzlich zu den hier aufgeführten Kosten kommen weitere Posten hinzu. Bspw. müssen jährliche Supportkosten für die eingesetzten Testwerkzeuge entrichtet werden. Die entwickelten Testskripte müssen auch einer Wartung unterzogen werden, welche ca. 10% vom Erstellungsbudget [77] ausmacht. Testmodellanpassungen können bisher nicht geschätzt werden, dafür fehlen die in der Praxis noch zu sammelnden Erfahrungswerte.

Kostenaufstellung pro Werkzeug	Kosten	Support-kosten (jährlich)
HPQC Lizenzkosten pro Workbench	2000 Euro pro Lizenz[103]	10.000 Euro
QTP pro Workbench	1250 Euro pro Lizenz	5000 Euro
MID GmbH (IBA) Modellierungswerkzeug	2800 Euro pro Lizenz[104]	1500 Euro
MBTSuite von Seppmed GmbH (Testfallgenerator)	7800 Euro pro Workbench[105]	2500 Euro
MBTSuite Beraterstandard bei Einführung (10–12	ca. 14400 Euro	

[102] Interviews mit Beratungshäusern (HAYS, MT AG und SAP AG), Standardstundensatz für Berater und Softwaretester.
[103] HP (enthalten sind die Kosten der Vertriebsabteilung für Lizenzen).
[104] Angebot eingeholt bei Seppmed GmbH – Hr. Dr. Beisser.
[105] Angebot eingeholt bei Seppmed GmbH – Hr. Dr. Beisser.

Manntage pro 1200 Euro)		
Summe	28.250 Euro	19.000 Euro

Tabelle 3: Kostenaufstellung Testwerkzeuge

Von dieser Berechnung ausgenommen sind der Aufwand für die Erstellung der Testfallmodelle und der administrative Aufwand, welcher getätigt werden muss, bevor ein Testskript zum Einsatz kommt. Für die hier betrachteten Testmodelle konnten folgende Zeitschätzungen vorgenommen werden:

Testmodell inklusive aller Teilmodelle	Stunden * Stundensatz	Kosten pro Testmodell
CFM-Kundensystem	16 Arbeitsstunden * 85 Euro	1360 Euro
Kreditsystem	12 Arbeitsstunden * 85 Euro	1020 Euro
Kassensystem	12 Arbeitsstunden * 85 Euro	1020 Euro
SAP-FI	32 Arbeitsstunden * 85 Euro	2720 Euro
Warenwirtschaftssystem	6 Arbeitsstunden * 85 Euro	510 Euro
Summe	78 Arbeitsstunden	6.630 Euro

Tabelle 4: Testfallmodellerstellung Kosten

Die Kosten für die Testmodellierung sind nur der erste Teil. Der zweite Teil muss in die Implementierung der Testskripte investiert werden. Allerdings hat dies den Vorteil, dass viele der hier entwickelten Funktionsbibliotheken wiederverwendbar sind. Die hier notwendigen Arbeitsschritte beinhalten die Modellierung der Testfälle, die Generierung und die Adaptierung.

Testskriptprogrammierung	Stunden * Stundensatz	Kosten
CFM-Kundensystem (20 Testskripte)	40 Arbeitsstunden * 85 Euro	3400 Euro
Kreditsystem (15 Testskripte)	15 Arbeitsstunden * 85 Euro	1275 Euro
Kassensystem (15 Testskripte)	15 Arbeitsstunden * 85 Euro	1275 Euro
SAP-FI (70 Testskripte)	60 Arbeitsstunden * 85 Euro	5100 Euro
Warenwirtschaftssystem (10 Testskripte)	6 Arbeitsstunden * 85 Euro	510 Euro
Summe 130 Testskripte	136 Arbeitsstunden	11.560 Euro

Tabelle 5: Die Kosten für die Erstellung der Funktionsbibliotheken

Die Vorgehensweise besteht darin, Funktionsbibliotheken zu erstellen und diese sukzessive zu erweitern. So können beliebig zusätzliche Softwaresysteme automatisiert werden. Der Stamm an Funktionen und Softwarekomponenten aller Art kann stetig zunehmen. So konnten jetzt mit 130 angefertigten Testskripten 384 Testfälle (IT-gestützter Zahlungsabwicklungsprozess) abgedeckt werden. Zusätzlich bieten diese Testskripte den Vorteil, mehrmals hintereinander und mit verschiedenen Parametern ausgeführt werden zu können.

Aktivitäten	Kosten
Testmodelle	6630 Euro
Testskripte	11560 Euro
Testwerkzeuge[106]	28250 Euro
Summe	46.440 Euro

Tabelle 6: Kosten initial für Testwerkzeuge und für die Fertigung von Testskripten und Testmodellen

Die Kosten für die Einführung der Testwerkzeuge inkl. Lizenzen sind einmalige Kosten. Allerdings müssen sie bei einer Neueinführung berücksichtigt werden. Diese Kosten würden bei einem neuen Projekt entfallen. Die Supportkosten sind jährliche Zusatzkosten und müssen umverteilt werden. Die Kosten für das Anfertigen der Testskripte und Modelle belaufen sich auf ca. 18.190 Euro.

5.5.2 Testkosten ohne MBT und Testautomatisierung

Die an der Evaluierung beteiligten Softwaresysteme unterstehen im täglichen Betrieb eigenständigen Organisationen und Verantwortlichkeiten unterschiedlicher Personen im Unternehmen. So konnte der Autor dieser Dissertation folgende Aufwände und Personen ermitteln, um einen regressiven Test aller Softwarekomponenten, welche am IT-gestützten Zahlungsabwicklungsprozesses beteiligt sind, zu ermöglichen.

Softwaresystem	Anzahl Tester	Stunden pro Person	Testdauer * Stundensatz	Kosten
CFM-Kundensystem	3	120 Arbeitsstunden	360 Stunden * 85 Euro	30.600,00 Euro
Kreditsystem	5	120 Arbeitsstunden	600 Stunden * 100 Euro	60.000,00 Euro
Kassensystem	5	120 Arbeitsstunden	600 Stunden * 75 Euro	45.000,00 Euro
SAP-FI	8	120 Arbeitsstunden	960 Stunden * 97 Euro	93.120,00 Euro
Warenwirtschafts-system	4	100 Arbeitsstunden	400 Stunden * 85 Euro	34.000,00 Euro
3000 Testfälle				262.720,00 Euro

Tabelle 7: Manuelle Testfallermittlung und Testausführung ohne Automatisierung

Die obige Tabelle beinhaltet die Aufstellung der Arbeitsstunden, welche aus folgenden Arbeitsschritten bestehen: Testfälle spezifizieren, Testfälle ausführen, Testausführungsergebnis auswerten und vergleichen (Erfolg/Fehler) und Suche nach Gründen für Fehlerwirkungen.

[106] Einmalige Investition.

Aus dieser Aufstellung ergeht noch keine Aussage bzgl. Testqualität bzw. Testfallqualität. Enthalten sind die Aufwände für die Testfallerstellung aus funktionalen Anforderungen sowie für die Ausführung der bislang bekannten Regressionstests. In den Arbeitsstunden enthalten sind außerdem noch Abstimmungsgespräche, die Fehlerbehandlung und die Beschaffung von Testdaten. Ein Arbeitstag besteht aus 8 Arbeitsstunden, wobei die Dauer der Testphase sich aus den jeweiligen Projektplänen ableiten lässt. Demnach kostet ein Testdurchlauf insgesamt ca. 262.720,00 Euro. In einem Jahr stehen aufgrund von Releasewechseln mindestens vier Regressionstestphasen an. Insgesamt benötigt das Testen dieser Geschäftsprozesse im Jahr ein Budget von ca. 1.050.880 Euro. Die hier ermittelten Zahlenwerke basieren auf zwei unterschiedlichen Datenbasen. Die in Tabelle 5 aufgestellten Kosten umfassen einen kompletten regressiven Test aller Testprozesse. Die Zahlen aus Tabelle 4 basieren allerdings nur auf dem hier behandelten Fallbeispiel. Deshalb müssen jetzt die Kosten noch einmal aufgesplittert werden, damit ein Vergleich dieser beiden Tabellen möglich wird.

Testskriptprogrammierung	Stunden * Stundensatz	Kosten
CFM-Kundensystem (32 Testfälle)	40 Arbeitsstunden * 85 Euro	3400 Euro
Kreditsystem (37 Testfälle)	80 Arbeitsstunden * 85 Euro	6800 Euro
Kassensystem (39 Testfälle)	70 Arbeitsstunden * 85 Euro	5950 Euro
SAP-FI (261 Testfälle)	160 Arbeitsstunden * 85 Euro	13.600 Euro
Warenwirtschaftssystem (15 Testfälle)	24 Arbeitsstunden * 85 Euro	2040 Euro
Summe 384 Testfälle	374 Stunden	31.790 Euro

Tabelle 8: Manuelles Testen (Regressionstest) des IT-gestützten Zahlungsabwicklungsprozesses

Die geschätzten Stunden basieren auf Untersuchungen und Beobachtungen im täglichen Tagesbetrieb. Der Autor dieser Dissertation führte dabei Analysen und Proben durch, um diese Zeiten zu ermitteln. Die Anzahl der Stunden (374) basiert hierbei ausschließlich auf der Zeit für das regressive Testen des Debitoren-Prozesses, nicht für den kompletten Regressionstest aller anderen Prozesse rund um B2B.

5.5.3 Testkosten mit MBT und Testautomatisierung vs. manuelles Testen – ein Vergleich

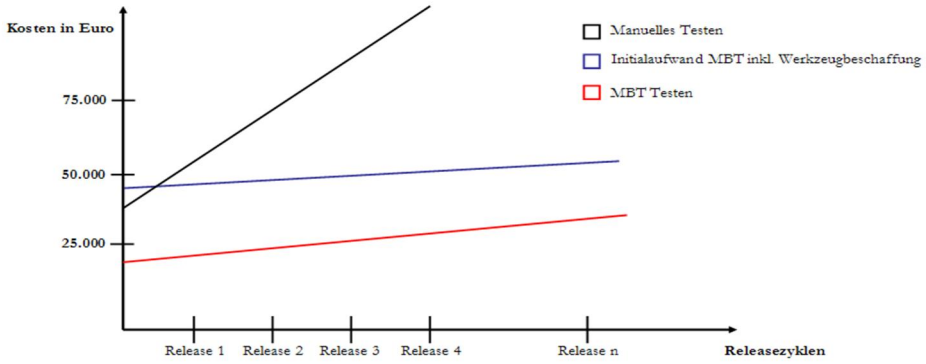


Abbildung 77: Kosten je nach Testmethode für den IT-gestützten Zahlungsabwicklungsprozess

Jetzt müssen die beiden ermittelten Werte miteinander verglichen werden, um zu zeigen, ab welchem Testlauf sich eine MBT-Methode am konkreten Beispiel lohnt.

Die Grafik verdeutlicht die Kostenverteilung, wobei der Initialaufwand bei der Ersteinführung berücksichtigt wurde. Das reine Erstellen und Programmieren der Testfälle aus den Modellen würde fast von Release zu Release stagnieren. Im ersten Lauf würden die Kosten für eine MBT-Methode die manuellen Testkosten übersteigen. Spätestens beim zweiten Release amortisiert sich die Investition. Was hier nicht in die Kalkulation aufgenommen wurde, ist die Testdauer beim automatisierten Testen. Im Vergleich mit einem manuellen Testansatz verringert sich der Zeitaufwand auf ca. 67% (zwei Drittel). Diese Zeitmessungen können sich je nach Komplexität und Variantenreichtum ändern. Gegenüber dem manuellen Test wird der Testprozess deutlich beschleunigt.

Neben dem manuellen Test wurde nach klassischer Art auch eine Testautomation als Testunterstützung eingesetzt. Wird die Testautomation zusätzlich zum hier betrachteten Kostenvergleich hinzugezogen, so verringert sich der Testaufwand insgesamt um ca. 50%. Mit anderen Worten, es kann, bereinigt hinsichtlich der MBT-Testmethode, der Zeitaufwand für das Testen um 50% reduziert werden. Das ist ein Erfolg.

5.5.4 Integration der MBT-Testmethode in eine Unternehmensorganisation

In diesem Unterabschnitt soll kurz dargestellt werden, wie eine erfolgreiche Einführung einer MBT-Methode in einer Unternehmensorganisation aussehen kann. In [122] wird die Investition in eine solche MBT-Methode als Aufwand und Nachteil beschrieben. Der Autor dieser Dissertation sieht dies eher als eine Anfangsinvestition, wie bspw. die Erweiterung der Fähigkeiten der Tester, um auch Modellierungen in BPMN vornehmen zu können. Das sollte nicht als Nachteil, sondern als eine anfängliche Investition, die sich langfristig gesehen rentiert, angesehen werden. Aufgrund dieser Investition ist es jetzt möglich, systematische und automatische Tests mit verhältnismäßig wenig Aufwand zu erreichen, was eine gute Softwarequalität nach sich zieht. Fallstudien über das modellbasierte Testen bestätigen dies [48], [122].

Förderlich für die erfolgreiche Einführung einer MBT-Testmethode ist eine in der jeweiligen Unternehmensorganisation gelebte Testkultur. Ein Vorgehen nach IT-Governance-Kriterien unterstützt hierbei die Einteilung der verschiedenen Rollen und sorgt gleichzeitig für ein Vier-Augen-Prinzip. Mit Governance-Prozessen in Unternehmen werden die Verantwortlichkeiten (Fachbereich, Tester, Entwickler, Projektmanager usw.) zwischen den agierenden Parteien deutlich abgegrenzt. Das Thema IT-Governance ist weit gefächert, daher reicht für die Dissertation ein kleiner Ausschnitt aus der IT-Tester-Rolle [68].

Das IT-Governance definiert für das Testen während einer IT-Entwicklungsphase 5 Quality Gates. Wobei die Gates mit Hilfe von „Checklisten“ überprüft werden müssen. Das Gate 1 erfordert die Prüfung der funktionalen Anforderung, das Gate 2 prüft die dazu passend von der Entwicklung geschriebene technische Umsetzung. Sind diese beiden Dokumente durch die Qualitätsprüfung gegangen, so erfolgt dann im Gate 3 die Abnahme des Modultests durch eine dritte, am Test unbeteiligte Person. Deren Aufgabe ist es, die von

der Entwicklung geschriebenen Testfälle zu analysieren und mit dem entwickelten Modul zu vergleichen. Ist auch diese Prüfung erfolgreich, erfolgt anschließend im Gate 4 ein Integrationstest aller am Test beteiligten Systeme. Hier werden im Speziellen die Integrationstests umgesetzt, auf Basis derer auch die End-to-End-Tests durchgeführt werden. Das Gate 5 ist das entscheidende Gate, welches vom Kunden durchgeführt wird. Hier werden die Kundenabnahmetests und die dazugehörigen Kriterien nicht nur definiert, sondern tatsächlich auch am Softwaresystem untersucht. Damit jede dieser verschiedenen im Prozess agierenden Personen ihre Aufgabe wahrnehmen kann, muss eine Abgrenzung her, genau diese bietet ein IT-Governance-Tool.

Bei der Integration einer MBT-Testmethode muss jetzt die Anforderung schon im Gate 1 in ein Testmodell überführt werden. So muss als Abnahmekriterium für das Gate 1 auch ein modelliertes Testmodell in BPMN hinzugefügt werden. Die IT-Governance-Rollen Modultester, Integrationstester und Kundenabnahmetester sorgen dafür, dass Verantwortlichkeiten auch klar definiert sind. Gleichzeitig aber erfordert dies von der Unternehmensorganisation, dass diese Prozesse gelebt werden und mit ihnen auch die verschiedenen Teststufen während eines Entwicklungs- oder Softwareprojektes.

Gleichzeitig müssen sowohl Tester und Fachbereiche sich für neue Themen öffnen, IT-Verantwortliche müssen ihre IT-Projekte besser planen, indem bspw. Tester frühzeitig in IT-Projekte eingebunden werden. Von den Testern erfordert dies Programmierfähigkeiten in Skriptsprachen wie VBA, vor allem müssen Programme so geschrieben werden, dass sie wiederverwendbar und leicht erweiterbar sind. Auch das stellt Tester vor neue Herausforderungen. Fachbereiche müssen sich viel mehr in die Modellierung einbinden und müssen in der Lage sein, auch eine fertig generierte Testsuite jederzeit ausführen zu können.

Abschließend muss festgehalten werden, dass auch die beste Testmethode ihren Sinn verliert, sollte eine Unternehmensorganisation nicht hinter dem Thema Testen stehen. Es erfordert viel Überzeugungskraft, das Thema Testen sowohl Entwicklern als auch Fachbereichen schmackhaft zu machen. Allerdings sind mit dem Effekt der Reduzierung von Fehlern in Produktivsystemen, der sich auch in der Praxis feststellen lässt, genug Argumente vorhanden, um das Testen im Unternehmen zu etablieren.

5.6 Zusammenfassung der erzielten Ergebnisse

Das Fallbeispiel zeigt, dass die Einführung dieser Methode mit einem Initialaufwand (Investition) verbunden ist, dass diese Methode sich aber schon nach wenigen Testläufen (Iterationen) amortisiert. Gleichzeitig erhöht die Methode schrittweise die Testabdeckung der Softwaresysteme, was sich unmittelbar positiv auf die Qualität der Softwaresysteme auswirkt.

Die Aufwände für die Wartung und die Pflege von Testfällen können nahezu auf null reduziert werden. Anders sieht es bei der Wartung der Testskripte aus, hier muss ein minimaler Aufwand betrieben werden, um die entwickelten Funktionsbibliotheken lauffähig zu halten. Erste Erfahrungswerte zeigen, dass im Durchschnitt 5% der Gesamtkosten auf die Wartung der entwickelten Testskripte entfallen. Damit die vorgestellte Methode erfolgreich umgesetzt werden kann, bedarf es neben den Testwerkzeugen auch einer strikten Versionierung der erstellten Testfallmodelle und Testskripte. Insgesamt schlägt der Autor dieser

Dissertation vor, die Einführung dieser Methode als ein Softwareprojekt zu betrachten. Die hier anzufertigen Testmodelle und Testskripte sollten dann einem regulären Softwarewartungszyklus unterzogen werden, somit kann sichergestellt werden, dass die Testskripte immer lauffähig bleiben.

Es konnte während der Evaluierung beobachtet werden, dass eine frühe Fehlererkennung und eine schnelle Fehlerbehandlung schon bei der Modellierung im Testmodell erfolgten. Die frühe Fehlererkennung trägt entscheidend dazu bei, Folgekosten aufgrund von Fehlerbehebungen zu minimieren.

Die Methode fordert die beteiligten Mitarbeiter heraus, so müssen sowohl Tester als auch Fachbereiche ihre gewohnten Rollen erweitern.

Der Tester muss neben dem „manuellen Testen“ jetzt in der Lage sein, Testfälle als Testmodelle mittels BPMN darzustellen und so zu formulieren, dass alle kritischen Pfade eines Softwaresystems abgedeckt sind. Zusätzlich muss er sich als Programmierer der Funktionsbibliotheken erweisen und methodische Programmierkenntnisse vorweisen. Das frühe Einbinden sowohl der Tester als auch der Fachbereiche in ein Entwicklungsprojekt kann als Erfolgsfaktor angesehen werden. Es entstehen Synergieeffekte, da Tester, Entwickler und Fachbereich gemeinsam das Testmodell verifizieren.

Dies fördert den Zusammenhalt in einer Unternehmensorganisation, was implizit die Zufriedenheit der Mitarbeiter erhöht.

Neben dem quantitativen Ergebnis der Verringerung des nötigen Testaufwands ergaben sich auch vielfältige qualitative Vorteile. Durch die Nutzung von Modellen und durch die bessere Rückverfolgbarkeit waren Regressionstests deutlich besser dokumentiert und die verschiedenen am Test beteiligten Parteien waren besser abgestimmt als zuvor, da das Testmodell eine gute Diskussionsgrundlage für alle Parteien bildet.

6 Zusammenfassung und Ausblick

Der hier beschriebene Ansatz integriert die bisherigen klassischen Testansätze und vergrößert sie durch die Erweiterung der BPMN-Spezifikation um eine Testspezifikation (inkl. Testelemente). Durch diesen Ansatz lassen sich beliebig verteilte BIS in ihrer Gesamtheit und End-to-End testen. Die BPMN-Spezifikation wurde hiermit um Testelemente erweitert, welche eine Modellierung in eine Testnotation erlauben. Wobei hier der Fokus auf den fachlichen Regressionstest gelegt wird. Eine weitere Erweiterung bezog sich auf die in der Praxis am Markt befindlichen Testmanagementwerkzeuge und auf die Funktion der Adaptierung. Somit konnte in dieser Dissertation gezeigt werden, wie auf der Basis einer BPMN-Testspezifikation Testfälle fachlich End-to-End modelliert sowie aus den BPMN-Modellen automatisiert Testfälle und Testskripte generiert und ausgeführt werden können.

Durch die Erweiterung können mit dem MBTSuite-Testfallgenerator jetzt aus BPMN-Testmodellen Testfälle generiert und interpretiert werden.

Gleichzeitig entstand ein Pool an Regressionstestfällen, welche bei jeder Softwareänderung bzw. -anpassung automatisiert ausgeführt werden müssen. Damit eine Automatisierung der Ausführung auf Basis der in BPMN erfassten Testschritte möglich wird, musste eine Adaptierung programmiert werden. So konnte erreicht werden, dass vom modellierten Testschritt zum Testskript eine Verbindung aufgebaut wird.

Durch die Involvierung der Fachbereiche in die Testfallmodellierung konnten Fehler beim Testfallentwurf frühzeitig erkannt und behoben werden. Auch konnten bis dato bestehende Testfälle auf Fehler und Konsistenz geprüft werden. Das Involvieren der Fachbereiche im Rahmen dieser Dissertation für die Ermittlung der Testanforderungen und für den Testfallentwurf kann als Erfolg bezeichnet werden. Auf Basis der erzielten Ergebnisse können zukünftig Schritt für Schritt weitere unternehmenskritische Prozesse nach dieser Methode getestet werden, so dass die Ergebnisse dieser Dissertation gleichzeitig ein „Best Practice“ für die zukünftige Zusammenarbeit bilden.

Ein weiteres Ergebnis dieser Dissertation ist, dass für die spezielle Modellierung von Testfällen mit BPMN Testprofile angelegt werden mussten, da BPMN speziell für die Geschäftsprozessmodellierung konzipiert wurde. Spezielle Elemente, welche für die Testfallmodellierung vonnöten sind, wie die Modellierung von Vor- und Nachbedingungen usw., wurden nicht berücksichtigt. Mit Hilfe von Testprofilen lassen sich diese Defizite kompensieren und jetzt auch für andere Testprojekte nutzen.

Jedes dieser am Fallbeispiel der Dissertation beteiligten Softwaresysteme in sich als komplette, eigenständige Softwarelösung in BPMN zu modellieren, ist eine sehr große Herausforderung. Die für das Fallbeispiel der Dissertation angefertigten Modelle dienten dabei schon als Grundstein.

Große verteilte Softwaresysteme enthalten fast zwangsläufig Fehler. Das Finden und Beheben dieser Fehler ist für einen großen Teil des Aufwandes beim Entwickeln von Softwaresystemen verantwortlich. Neben anderen Techniken ist das Testen von Software eine der Hauptaktivitäten zum Finden von Fehlern. Egal welches Software-Vorgehensmodell für

das Entwickeln von Software genutzt wird, das Testen sollte Bestandteil eines jeden Vorgehensmodells sein. Die Motivation, diese Dissertation zu verfassen, entstand durch die Herausforderungen des täglichen IT-Betriebs von verteilten BIS in der Praxis. Durch die immer kürzer werdenden Produktauslieferungszyklen [157] und durch die wachsenden Anforderungen des Tagesbetriebes wurde ein Umdenken beim Testen solcher komplexen und verteilten Softwaresysteme nötig.

6.1 Was wurde durch diese Dissertation erreicht?

Insgesamt mussten im Rahmen dieser Dissertation zwei wesentliche Komponenten erweitert werden. Die eine ist die BPMN-Spezifikation der Testelemente, welche eine Testmodellierung in einer Testnotation ermöglichen soll. Als Zweites wurde das Testmanagementwerkzeug um die Funktion der Adaptierung erweitert, welche eine Referenzierung auf implementierte Testskripte erlaubt. Beide Erweiterungen machen erst das Funktionieren der hier vorgestellten Methode möglich. Die Basis für Adaptierungen bieten die angefertigten Funktionsbibliotheken. Diese Bibliotheken dienen jetzt als Standardset eines Testmanagementwerkzeugs, um ERP-Systeme zu testen. Eine dritte, nicht unwesentliche Komponente ist die Verbesserung der Zusammenarbeit mit einzelnen Fachbereichen, welche über die konkreten Geschäftsprozessabläufe verfügen, um sie frühzeitig in Testprojekte zu involvieren. BPMN als Modellierungsnotation trug maßgeblich dazu bei, die Akzeptanz zu erhöhen. Eine letzte Komponente ist die Einführung von Namenskonventionen in Testprojekten. Diese bilden den Grundstein und die Basis für das Funktionieren der hier vorgestellten Methode.

Erprobt wurde die hier vorgeschlagene Methode im Rahmen eines Testprojektes unter Praxisbedingungen. Insgesamt konnte gezeigt werden, dass der Testaufwand auf ca. 50% reduziert werden kann, bei gleichzeitiger Erhöhung des Testabdeckungsgrads am Fallbeispiel um 100%. Der Abdeckungsgrad ist abhängig von der ausgewählten Generierungsstrategie. Zusammen mit einem Fachbereich kann einfach festgelegt werden, welche Geschäftsprozesspfade in einem Test berücksichtigt werden müssen und welche nicht. Dadurch kann der Abdeckungsgrad zusammen mit einem Fachbereich gesteuert werden. Das Ziel war es, eine 100%ige Prozessüberdeckung für das Fallbeispiel des IT-gestützten Zahlungsabwicklungsprozesses im Regressionstestpool zu haben. Dieses Ziel konnte im Rahmen der Dissertation erreicht werden.

Ein Seiteneffekt, welcher durch die Dissertation beobachtet werden konnte, ist das Umdenken der beteiligten Parteien in Bezug auf das Thema Testen. Das Testen ist zwar ein fester Bestandteil im Rahmen von Softwareentwicklungsprozessen, wird aber gerne vernachlässigt [1], [30]. Durch die gemeinsame Testfallmodellierung durch Fachbereiche, Tester und Entwickler konnte ein Umdenken bezüglich des Berufsbildes des Testers beobachtet werden. Fachbereiche konnten unter Praxisbedingungen wahrnehmen, wie eine konstruktive und analytische Qualitätssicherung in Kombination mit dem Einsatz von Test- und Modellierungswerkzeugen dazu beitragen kann, nicht nur die Fehler in Produktivumgebungen zu verringern, sondern auch durch aktive Mitarbeit bis dato gelebte Geschäftsprozesse zu verbessern. Fehler, die auch auf Missverständnisse bei der Anforderungserfassung zurückzuführen sind, konnten in der Modellierungsphase frühzeitig erkannt und geklärt werden.

Auch konnte bei der Testfallmodellierung im Rahmen dieser Dissertation eine Vielzahl von Softwarepfaden identifiziert werden, zu welchen bisher keine textuellen Testfälle beschrieben waren. Die zukünftige Zusammenarbeit der einzelnen Parteien wird zusätzlich durch fest definierte und unternehmensweit geltende Namenskonventionen bei der Testfallbenennung unterstützt.

6.2 Kritische Würdigung

Die Arbeit beschreibt, wie eine Methode zum modellbasierten Testen von End-to-End-Geschäftsprozessen erfolgen kann. Dazu hat der Autor sowohl die Problemstellung in der Praxis beschrieben als auch Konzepte beispielhaft dargestellt. Allerdings fehlen eine detaillierte Darstellung dieser Konzepte und deren Abgrenzung. Auch existierten zum Zeitpunkt der Anfertigung dieser Dissertation wenige (vier) wissenschaftliche Ausarbeitungen zum Schwerpunkt dieser Arbeit.

Die Erweiterung und die Realisierung dieser Methode mittels BPMN erfolgten nur mit der Begründung der Benutzerfreundlichkeit, jedoch wären hier vielmehr eine scharfe Abgrenzung zu UML und gleichzeitig eine tiefgreifende Erläuterung der Erweiterung vorteilhafter gewesen. Die hier benutzten IT-Begrifflichkeiten aus dem Themenkomplex des IT-Qualitätsmanagements setzte der Autor dieser Dissertation meist als bekannt voraus, ohne explizit darauf hinzuweisen.

Die hier vorgestellte Methode wurde in der Praxis evaluiert und getestet. Die an der Evaluierung beteiligten Applikationen wurden sehr ausführlich beschrieben. Die Methode lässt sich aber nicht auf mobile Applikationen übertragen. Mobile Applikationen waren zu keiner Zeit Bestandteil dieser Dissertation. Auch muss eine genaue Untersuchung erfolgen, mit welchen Werkzeugen die BPMN-Erweiterung so wie hier beschrieben und dargestellt funktionieren kann. Das Coding ist sehr kompliziert und muss in einem weiteren Schritt noch organisiert und strukturiert werden. Eine Übertragung auf andere Modellierungswerkzeuge wurde nicht betrachtet, kann aber in einer weiteren Arbeit untersucht werden.

6.3 Ausblick

Im Rahmen dieser Dissertation wurde der IT-gestützte B2B-Prozess mit der modellbasierten Testmethode evaluiert. Die Basis für das MBT mit BPMN ist mit dieser vorgestellten Methode gelegt, weitere Arbeiten im Umfeld des modellbasierten Testens mit BPMN können darauf aufbauend die Integration der verschiedensten Werkzeuge verbessern. So ist die technische Lösung mit Hilfe von IBA und mit dem speziellen Anlegen von Testprofilen zwar heute eine gute Alternative zu UML. Allerdings erfordert das Modellieren und Transferieren des Testmodells mit einem Testfallgenerator spezielle Konfigurationen, welche durch einen Softwaretest und durch Modellierungsexperten angelegt werden müssen. Auch das Exportieren in die MBTSuite ist nicht frei von Problemen, so müssen auch hier Konfigurationsdateien dafür sorgen, dass das Modell korrekt von IBA zur MBTSuite exportiert bzw. importiert wird.

Auch ist die Lösung mit der Adaptierung von Testskripten zu korrespondierenden Funktionen in der Funktionsbibliothek entwickelt und erprobt. Wie kann diese Adaptierung

bspw. direkt in einen Testfallgenerator eingebaut werden? Zukünftige Arbeiten können sich darauf konzentrieren, eine „all in one“-Lösung anzustreben, so dass sowohl die Modellierung in BPMN als auch die Testfallgenerierung aus einer Softwarelösung erfolgen.

Auch die im Rahmen dieser Dissertation entwickelte Adaptierung von manuell generierten Testfällen zu ausführbaren Testskripten kann verbessert werden. Bspw. könnte ein Add-in entwickelt werden, welches die Adaptierung dieser Testfälle IT-gestützt ohne Zwischenschritte oder Medienbrüche ermöglicht.

Ein Folgeprojekt, welches im Rahmen der Evaluierung der hier entwickelten Methode entstand, ist das Verfeinern der Zusammenarbeit zwischen Fachbereichen, IT-Experten (Entwicklern), Business-Analysten und Softwaretestern. In diesem Folgeprojekt sollen jetzt Untersuchungen dazu durchgeführt werden, welche Softwaresysteme sich zusätzlich dazu eignen, modellbasiert getestet zu werden, und wie Voraussetzungen geschaffen werden können, um die Einführung und die Etablierung zu unterstützen.

Eine weitere sinnvolle Ergänzung bezieht sich auf das Testframework, es bietet zwar beim Testen eine Unterstützung, kann aber besser vereinheitlicht und erweitert werden. Der Grundstein ist mit der Generierung von Testfällen und mit der Einbindung von Funktionsbibliotheken mit „Standardangriffen“ (Ausführungen von vorher programmierten Funktionsbausteinen) gelegt.

Die in dieser vorliegenden Arbeit im Verbund mit der Testmethode vorgestellten Testwerkzeuge können sicherlich noch weiterentwickelt und erweitert werden. Wichtiger wäre aber eine Überprüfung an ähnlichen Softwaresystemen und in anderen Unternehmen und Branchen. Die vorgestellte Methode wurde ausschließlich an Software aus der Handelsbranche ausprobiert und untersucht.

Durch die in dieser Arbeit erreichten Ziele wird ein Ersteller von Testmodellen in die Lage versetzt, mit den definierten Namenskonventionen und mit den Adaptierungen der Testfälle von zukünftigen Anforderungen fertig automatisierte Testskripte zu erstellen. Weiterhin können die modellierten Testaktivitäten und deren Testdaten auf Verhaltensvorgaben geprüft werden.

Der Modellierer wird bei der Testanforderungsmodellierung durch die jeweiligen Fachbereiche unterstützt, die sicherstellen, dass alle notwendigen Aspekte des Systems berücksichtigt werden und zudem eine sinnvolle Reihenfolge bei der Modellerstellung eingehalten wird. Insgesamt kann damit ein Qualitätsniveau für die zu generierenden Testfälle erzielt werden.

Anhang

Fragenkatalog I (siehe beigefügte CD)

Fragenkatalog II (siehe beigefügte CD)

Testmodell CFM (siehe beigefügte CD)

Testmodell Kassensystem (siehe beigefügte CD)

Testmodell Kreditsystem (siehe beigefügte CD)

Testmodell Warenwirtschaftssystem (siehe beigefügte CD)

Testmodell SAP-FI-Zahlprozess (siehe beigefügte CD)

Testmodell SAP-FI-Mahnprozess (siehe beigefügte CD)

Funktionsbibliotheken (siehe beigefügte CD)

Adaptierungsskript (siehe beigefügte CD)

Literaturverzeichnis

[1] Utting, M., Legeard, B.: „Practical Model-Based Testing: A Tools Approach“, 2007

[2] A. Metzger: „Erfahrungen mit modellzentriertem Testen in der Validierung komplexer sicherheitskritischer Systeme“, 2009 TAV, http://www.gm.fh-koeln.de/~winter/tav/html/tav29/TAV29P03Metzger_seppmed.pdf

[3] J. Bergmann: „Testautomatisierung Kosten und Nutzen“, 2009, Ausgabe 1-2009, http://www.software-quality-lab.com/uploads/media/SWQL-Newsletter-200903_-_Testautomatisierung_-_Kosten_und__Nutzen_01.pdf

[4] F. Steimann, D. Keller: „Moderne Programmiertechniken“, Vorlesungsmanuskript 2012/2013; Kapitel „Testen“

[5] O. J. Dahl, E. Dijkstra, C.A.R Hoare: Structured Programming. New York: Academic Press 1972

[6] P. Stahlknecht, U. Hasenkamp: „Einführung in die Wirtschaftsinformatik“, Axel Springer Verlag 2004

[7] A. Spillner, A. Lenz: „Softwaretest: Aus- und Weiterbildung zum Certified Tester; Foundation Level nach ISTQB-Standard“, dPunkt Verlag 2005

[8] SQS, Quality, „SAP-Testautomation von HR-Anwendungen“, Fachzeitschrift SQS Quality 2008, Ausgabe 8

[9] SQS, Quality, „SAP-Testautomation bei Mercedes Benz“, Fachzeitschrift SQS Quality 2008, Ausgabe 8

[10] P. Liggesmeyer: „Software-Qualität. Testen, Analysieren und Verifizieren von Software“, Spektrum Akademischer Verlag 2002

[11] A. Pretschner et al.: „One evaluation of model-based testing and its automation“, Whitepaper, Veröffentlichung 2005

[12] S. Wieczorek, A. Stefanescu: „Improving Testing of Enterprise Systems by Model-Based Testing on Graphical User Interfaces“, Whitepaper 2010

[13] M. Mlynarski: „Holistic Model-Based Testing for Business Information Systems“, Whitepaper 2010

[14] B. Henderson: „Object-Oriented Metrics, Measures of Complexity“, Prentice-Hall 1996

[15] D. W. Hoffmann: „Software-Qualität“, Springer Verlag Vieweg, 2. Auflage, 2008

[16] S. Kleuker: „Grundkurs Software-Engineering mit UML“, 2008

[17] Beulen, D.: „Ansätze des Modellbasierten Testens“, veröffentlicht 2009, Universität Paderborn

[18] M. Friske: „Testfallerzeugung aus Use-Case-Beschreibungen“, FU Berlin, 2004

[19] J. Grabowski et al.: „An introduction to the testing and test control notation (TTCN-3)“, 2008

[20] J. Peleska et al.: „Automated test case generation with SMT-solving and abstract interpretation“, Universität Bremen 2011

[21] G. Stefania et al.: „Formal Test-Case Generation for UML Statecharts“, 2004

[22] A. D. Neto et al.: „A Survey on Model-based Testing Approaches: A Systematic Review, Siemens Corporate Research”, Whitepaper 2007

[23] S. Monalisa, M. Rajib: „Automatic Test Case Generation from UML Models“, Whitepaper 2007

[24] S. Monalisa, M. Rajib: „System Testing using UML Models“, Whitepaper, 2007

[25] D. Sokenou: „UML-basierter Klassen- und Integrationstest objektorientierter Programme“, Dissertation, 2006

[26] T. Eckardt, M. Spijkerman: „Modellbasiertes Testen auf Basis des fundamentalen Testprozesses“, Beitrag zur TAV 28 in Dortmund, 2009

[27] S. Kansomkeat et al.: A Comparative Evaluation of Tests Generated from Different UML Diagrams. Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2008), S. 867–872, Phuket 2008

[28] K. Hyungchoul et al.: Test Cases Generation from UML Activity Diagrams. SNPD (3) 2007, 556–561

[29] Booch: „The Unified Modeling Language User Guide“, Addison-Wesley Object Technology, 2005

[30] H. Götz et al.: „Modellbasiertes Testen“, „iX-Studie Modellbasiertes Testen“, Heise Zeitschriften Verlag, Hannover 2009

[31] B. Güdali et al.: „STARTHILFE FÜR MODELLBASIERTES TESTEN: ENTSCHEIDUNGSUNTERSTÜTZUNG FÜR PROJEKT- UND TESTMANAGER“, in: OBJEKTspektrum 03/2010

[32] Datenverarbeitungszentrum Mecklenburg-Vorpommern: „Ausgewählte Modellierungsnotationen“, 2011, http://www.econbiz.de/archiv/ka/uka/information/eorganisation_service_market1.pdf

[33] M. Michalis, Optimierung des Zahlungsverkehrs mit SAP In-House Cash, SAP-PRESS, Auflage 1, 2009

[34] S. Neil: „Safety Critical Computer Systems“, Addison-Wesley Longman Publishing 1996

[35] MATLAB/Simulink/Stateflow (Product Information): The MathWorks Inc., www.mathworks.com/products

[36] B. Rumpe: „Agile Modellierung mit UML-Codegenerierung, Testfälle, Refactoring“ Springer Verlag, 2. Auflage, 2011

[37] Pretschner, A.: „Zur Kosteneffektivität modellbasierten Testens“, Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme II, Braunschweig 2006

[38] M. Lim, S. Sadeghipour: „Automatisierte Strukturtests in der modellbasierten Entwicklung“, 2006

[39] H. Schlingloff, M. Conrad, H. Dörr, C. Sühl: Modellbasierte Steuergerätesoftwareentwicklung für den Automobilbereich, in: GI-Tagung „Automotive Safety and Security 2004 – Sicherheit und Zuverlässigkeit für automobile Informationstechnik“ Stuttgart 2004

[40] H. Schlingloff et al.: „Qualitätssicherung lebenswichtiger Steuergerätesoftware“, FU Berlin, Whitepaper, Veröffentlichung 2008

[41] Eclipse Foundation: „Eclipse Modeling Framework“, http://www.eclipse.org/emf/, 2012

[42] D. Tefkat: „UML-basierte Testfall- und Systemmodelle für die Eisenbahnleit- und -sicherungstechnik, http://www.dstc.edu.au/Research/Projects/Pegamento/tefkat/, Dissertation, Veröffentlichung 2008

[43] M. Friske, H. Schlingloff, S. Weißleder: Composition of Model-based Test Coverage Criteria. In: MBEES 2008 - Model-Based Development of Embedded Systems, 7.4. - 10.4.2008, Dagstuhl; IPS-Bericht 2007-01, TU Braunschweig, S. 87–94

[44] OMG-Spezifikation: „Object Management Group. (MDA) Guide Version 1.0.1“, http://www.omg.org/mda/, 2003

[45] M. Friske, H. Schlingloff: Generierung von UML-Modellen aus formalisierten Anwendungsfallbeschreibungen, in: MBEES 2007 - Model-Based Development of Embedded Systems, 15. - 18.01.2007, Dagstuhl; Informatik-Bericht 2007-01, TU Braunschweig, S. 113–132

[46] A. Spillner et al.: „Praxiswissen Softwaretest – Testmanagement“, dPunkt Verlag, 2007

[47] J. Link: „Softwaretest mit JUNIT“, dPunkt Verlag 2005

[48] SOFTWARE ACQUISITION GOLD PRACTICE: MODEL-BASED TESTING, The Data and Analysis center for Software (DACS), (http://www.goldpractices.com/practices/mbt/index.php)

[49] N. Hartmann: „Automation des Tests eingebetteter Systeme am Beispiel der Kraftfahrzeugelektronik“, http://digbib.ubka.uni-karlsruhe.de/volltexte/1642001, 2001

[50] O. Gietelink et al.: „Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations“, 2006, http://www.dcsc.tudelft.nl/~bdeschutter/pub/rep/05_009.pdf?origin=publication_detail

[51] C. Wilcock et al.: „An Introduction to TTCN-3“, Wiley & Sons, 2. Auflage, 2005

[52] J. Peleska, et al.: „Automated test case generation with SMT-solving and abstract interpretation“, Whitepaper, Universität Bremen 2011

[53] K. Mewes: „Domain-specific modelling of railway control systems with integrated verification and validation“, Dissertation, http://www.informatik.uni-bremen.de/~kirsten/thesis.pdf

[54] S. Kelly, J.-P. Tolvanen: „Domain-Specific Modeling: Enabling Full Code Generation, Hoboken“, New Jersey: Wiley-IEEE Computer Society Press, 2008

[55] Metacase: http://www.metacase.com/products.html., 2012

[56] S. Weißleder: „Test models and coverage criteria for automatic model-based test generation with uml state machines“, Dissertation, http://model-based-testing.de/data/weissleder_phd_thesis.pdf

[57] S. Ranise, C. Tinelli: „Satisfiability modulo theories“, Magazine, Communication of the ACM, Volume 54, S. 69–77, September 2011

[58] ARINC SPECIFICATION: „Avionics Application Software Standard Interface, Part 3 – Conformity Test Specification“, Springer Verlag, 2007

[59] Myers, Glenford: „Methodisches Testen von Programmen“, Wissenschaftsverlag, Oldenbourg 2001

[60] M. Conrad: „Auswahl und Beschreibung von Testszenarien für den modellbasierten Test eingebetteter Software im Automobil“, Dissertation, http://swt.cs.tu-berlin.de/publ/diss.html, 2004

[61] Pretschner et al.: „Methodological Issues in Model-Based Testing“, Whitepaper, http://www22.informatik.tu-muenchen.de/fileadmin/papers/methodology05.pdf, 2004

[62] M. Broy et al.: „Model-Based Testing of Reactive Systems, Advanced Lectures“, http://drops.dagstuhl.de/opus/volltexte/2006/585/pdf/05402.Manifest_SE_Dagstuhl_IS.585.pdf, 2005

[63] M. Utting, B. Legeard: „Practical Model-Based Testing - A Tools Approach“, Morgan Kaufmann, 2006

[64] H. Balzert: „Lehrbuch der Software-Technik Band 2“, Spektrum Verlag, 2002

[65] B. Meyer: „empirical software engineering and verification“, Springer Verlag, Berlin, 2012

[66] M. Glenford et al.: „The Art of Software Testing“, John Wiley & Sons, 2004

[67] K. Ebanhesaten, FUH, Masterthesis, Testautomatisierung in SAP Systemen, https://www.fernuni-hagen.de/ps/arbeiten/ebanhesatenkhalid.shtml, 2011

[68] IT-Governance Template: Quelle: http://www.itgi.org/Template_ITG164fc.pdf?Section=About_IT_Governance1&Template=/ContentManagement/ContentDisplay.cfm&ContentID=14529

[69] G. E. Thaller: „Software-Test: Verifikation und Validation“, Heise Verlag, 2002

[70] S. Metsker: „Design Pattern Java Workbook“, Pearson Education, 2002

[71] D. Faught: „In information resource for software testers“, http://www.testingfaqs.org, 2012

[72] M. Aberdour: „Open source tools for software testing professionals“, 2012

[73] V. Knollmann: „UML-basierte Testfall- und Systemodelle für die Eisenbahnleit- und -sicherungstechnik“, Volker Knollmann, Dissertation erschienen Dezember 2007

[74] M. Pol et al.: „Management und Optimierung des Testprozesses: ein praktischer Leitfaden für das Testen von Software, mit TPI und TMap“, Addison-Wesley 2001

[75] T. J. Ostrand: „Using data flow analysis for regression testing“, Whitepaper: http://books.google.de/books?id=IvxVAAAAMAAJ&q=T.J.Ostrand:+%E2%80%9EUsing+data+flow+analysis+for+regression+testing&dq=T.J.Ostrand:+%E2%80%9EUsing+data+flow+analysis+for+regression+testing&hl=de&sa=X&ei=vhorU7-xM8GOtQbFk4H4DQ&ved=0CDUQ6AEwBA

[76] M. Tiede „Modellbasiertes Testen“, 2010

[77] K. Ebanhesaten, C. Stenhorst: „Gefahren und Risiken von Testautomation in großen Softwareprojekten“, 2011 HMD Verlag, Heft 278

[78] D. Neto: „A Survey on Model-based Testing Approaches“, 1 ACM Workshops on Empirical Assessment of Software Languages and Technologies, WEASELTech, 07, 2007, S. 31–36

[79] M. Jeckle: „UML 2 – glasklar“, Hanser Verlag 2004

[80] H.G. Gross: „testing and the UML – A Perfect Fit“, Cambridge University Press 2005

[81] HL7: „Business Continuity” http://www.hl7.org/

[82] IHE International: „Integrating the Healthcare Enterprise“, http://www.ihe.net, 2008

[83] OMG: UML, http://www.uml.org, 2008

[84] M. Wagner: Praxisleitfaden zur Berechtigungsprüfung im SAP, Online: http://www.mariewagener.de/files/active/0/Praxisleitfaden_Berechtigungspruefung_CO.pdf

[85] D. Kreische: „Modellbasiertes Testen“, Quelle: www.mid.de, 2010

[86] TNG: Test Next Generation Medical Systems EUREKA Project, 4053 RETEMES, 2009

[87] Galileocomputing: http://www.galileocomputing.de/download/dateien/734/galileocomputing_uml2null5.pdf, 2012

[88] D. Stiefel: „Wettkampf der Modellierungsmethoden“, Fachartikel im SAPERION BLOG, 2010

[89] Statistik: „www.bpm.netzwerke.de“, 2008

[90] Status Quo BPM, 2009

[91] M. Wolters, M. Kaschny: „Geschäftsprozessmodellierung in KMU“, 2010

[92] T. Allweyer, P. Loos: „Process-Orientation and Object-Orientation – An approach for integration UML and eventdriven Process Chains (EPC)“, 1998

[93] B. Oestereich: „Objektorientierte Softwareentwicklung: Analyse und Design mit der UML 2.0“, Oldenbourg Verlag, 1998

[94] OASIS: „Web Services Business Process Execution Language Version 2.0“, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel, 2007

[95] Oracle: „Oracle BPEL Process Manager“, http://www.oracle.com, 2012

[96] C. Rupp: „Requirements-Engineering und -Management“, 2004

[97] van der Aalst et al.: „Workflow Patterns in Distributed an Parallel Databases“, 2003

[98] S. Gnesi et al.: „Formal Test-Case Generation for UML Statecharts, in: Engineering of Complex Computer Systems“, Dissertation, http://www.informatik.uni-trier.de/~ley/pers/hd/g/Gnesi:Stefania, 2004

[99] C. Sun: „A Transformation-Based Approach to Generating Scenario-Oriented Test Cases from UML Activity Diagrams for Concurrent Applications“, in: Proceeding COMPSAC '08 Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference Pages 160–167

[100] K. Hyungchoui et al.: „Test Cases Generation from UML Activity Diagrams“, 2007

[101] P. Baker, et al.: „Model Driven Testing – Using the UML Testing Profile.“ Springer Verlag, 2008

[102] Ru. D. Zhen: „Model Driven Testing – with UML“, http://www.cskent.ac.uk/projects/kmf/mdaworkshop/submissions/Dai.pdf, 2004

[103] B. Holzer: „Modellgetriebene Softwareentwicklung – MDA und MDSD in der Praxis“, ISBN 978–3–939084– 11–2, 2007

[104] S. Beydeda, et al.: „Model–Driven Software Development“, ISBN 03–540–25613–X, 2005

[105] BPMN 2.0:Tutorial Kompakte Einführung in die BPMN 2.0, www.bpmn.de

[106] J. Mendling et al.: „In: Gesellschaft für Informatik (GI) e.V. (Hrsg.): Informationssystem Architekturen, Wirtschaftsinformatik Rundbrief der GI Fachgruppe WIMobIS, Jahrgang 10, Heft 2“, 2003

[107] H. Störrle et al.: „EPK 2006 - 5. Workshop der Gesellschaft für Informatik e.V. Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten“, 2006

[108] T. Griebel: Diplomarbeit zum Thema: Generierung von Testfällen aus UML; Universität Stuttgart, 2012

[109] C. Krzysztof, H. Simon: „Classification of Model Transformation Approaches. In: OOPSLA 2003 Workshop on Generative Techniques in the context of Model Driven Architecture“, 2003

[110] S. Shane, K., Wojtek: „Model Transformation: The Heart and Soul of Model-Driven Software Development. In: IEEE Software 20 “, 2003

[111] J. Warmer, A. Kleppe: „The Object Constraint Language: Getting Your Models Ready for MDA“, 2003

[112] R. V. Binder: „Testing Object-Oriented Systems: Models, Patterns, and Tools. Object TechnologySeries“, 2003

[113] Automotive SPICE – www.sogeti.com

[114] D. Tefkat: http://www.dstc.edu.au/Research/Projects/Pegamento/tefkat/

[115] ATZ: www.ATZonline.de, 2009

[116] B. Güdali: „Wann lohnt sich MBT?“ http://cdn1.hlmc.de/tl_files/mbtconf/site/Vortraege/2.%20Konferenztag/Forum%203/Baris%20Gueldali.pdf, 2010

[117] W. Meyer, L. Bichler: MBT bei Infotainment, http://subs.emis.de/LNI/Proceedings/Proceedings210/P-210.pdf, 2006

[118] Marko Graf, Matthias Scholze, Mario Friske: „Anforderungen mit HPQC verwalten“, iX Publikation, 2011

[119] O. Armbrust et al.: „Probleme etwa bei der Automatisierung von Softwaretests“, http://publica.fraunhofer.de/documents/N-24381.html, 2004

[120] K. C. Laudon et al.: „Wirtschaftsinformatrik“, Springer Verlag, Volume 49, 2007

[121] Seppmed GmbH: „Modellzentriertes Testen“, http://www.seppmed.de/modellzentrierterTest.187.0.html, 2007

[122] A. Hartmann, „AGEDIS“. Final Project Report, Feb. 2004, http://www.agedis.de/downloads.shtml

[123] B. Boehm: „Software Engineering Economics“, Pearson Education, 1981

[124] K. Beck et al.: „Extreme Programming“, Springer Verlag, 2. Edition, 2000

[125] Siedersleben: „Moderne Software Architektur“, dPunkt Verlag, 2004

[126] C. Siedentop: „Testautomatisierung – Konzepte und Praxis in JUNIT 4“, Open Source Press, 2008

[127] D. Majer: „Unit Test in ABAP“, dPunkt Verlag, 2008

[128] J. Link: „Unitest in Java“, dPunkt Verlag, 2002

[129] G. Greiter: „Test Automation Tools“, http://greiterweb.de/spw/test_werkzeuge.htm, 2011

[130] M. Fowler: „Refactorings – Improving the Design of Existing Code“, Addison Wesley Verlag, 1999

[131] IBM, TestCoductor Tutorial, http://publib.boulder.ibm.com/infocenter/rhaphlp/v7r6/index.jsp?topic=%2Fcom.ibm.rhp.integ.testingtools.doc%2Ftopics%2Frhp_r_dm_vendor_doc_testing.html

[132] Bundesministerium des Inneren: Entwicklungsstandard für IT Systeme des Bundes, Vorgehensmodell. http://www.v-modell.iabg.de/. Letzter Zugriff: April 2011. (Zitiert auf den Seiten 16 und 169.)

[133] R. Seidl: „Basiswissen Testautomatisierung - Konzepte, Methoden und Techniken“, dPunkt Verlag, 2007

[134] IEEE 610: IEEE Std. 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, IEEE Press, New York, 1990

[135] G. Rothermel, M. J. Harrold: „Analyzing Regression Test Selection“, Published in: Journal IEEE Transactions on Software Engineering archive, Volume 22 Issue 8, August 1996, Page 529–551

[136] J. Hartmann, D. Robson: „Techniques for Selective Revalidation“, http://pi.informatik.uni-siegen.de/stt/23_4/01_Fachgruppenberichte/TAV/P6Sneed2TAV20.pdf

[137] H. Sneed, M. Winter: „Testen objektorientierter Software“, Hanser Verlag, 2002

[138] P. Haberl, A. Spillner: „Softwaretest in der Praxis – Umfrage 2011“, dPunkt Verlag 2011

[139] HPQC Guide und Dokumentation, Version 2011

[140] SAP Deutschland, Journal zum Schutz von IT Investitionen, http://sap-cio.de/wp-content/uploads/2012/08/Loesung_im_Detail_SAP_Enterprise_Support_50090388.pdf, 2012,

[141] Conformiq Tool Guide, Version 2012

[142] B. Urban: „Debitoren und Kreditorenbuchhaltung“, HAUFE Verlag 2010

[143] K. Franz: „Handbuch zum Testen von Web-Applikationen“ Springer 2007

[144] D. Jessen: SAP Integration X4, 2009

[145] Dustin et al.: „Automated Software Testing: Introduction, Management, and Performance“, Addison Wesley 1999

[146] Rally: http://www.rallydev.com

[147] VersionOne: http://www.versionone.com

[148] ScrumWorks: http://danube.com/scrumworks

[149] K. Pohl, C. Rupp: „Basiswissen Requirement Engineering”, Third International Conference 2010

[150] D. W. Hoffmann: „Software-Qualität“, Springer-Verlag, 2010

[151] T. Allweyer: BPMN, eine Sprache auch für das Business, 2 Auflage, Books on Demand, 2009

[152] J. Freund et al.: Praxishandbuch BPMN, 2012

[153] T. Allweyer: BPMN, http://www.bpmnbuch.de, 2009

[154] M. Moser: „Wann lohnt sich Testautomatisierung“, www.qfs.de, Quality First, 2007

[155] Michael P. Papazoglou: Web Services: Principles and Technology. Prentice Hall, Essex 2007

[156] M. Biss: „Markenführung in B2B“, Diplomarbeit GRIN Verlag, 2005

[157] G. Tochtrop: „Testexperte als Anforderungsmanager“, Softwaretechnik-Trends 2008

[158] A. Kurtz: Handelsjournal, „Trendsetter im Handel“, Ausgabe 11, 2013

[159] P. Audon Consultants: „Marktstudie zu Software-Tests und Qualitätsmanagement“, 2011

[160] M. J. Wieczorek et al.: „Test von Software“, Springer Link (Xpert Press), 2003

[161] HP Deutschland – QTP Guide, Version 2011

[162] OMG. Business Process Model and Notation (BPMN) Version 2.0, 2011. (Zitiert auf den Seiten 10, 15, 16, 18, 19 und 39)

[163] OMG. Meta Object Facility (MOF) Core Specification, 2011 (Zitiert auf Seite 17)

[164] OMG. OMG Unified Modeling Language (OMG UML), Superstructure Version 2.4.1, 2011. (Zitiert auf Seite 19)

[165] Paper: Sahar Yousefi Barforooshi et al.: Improvement of Test Management in BPMN with ARIS, Conference: International Conference on Electronic Computer Technology – ICECT, 2010

[166] Paper: A. Jimenez-Ramirez et al.: Contract-based test generation for data flow of business processes using constraint programming, ICAART 2014 - Proceedings of

the 6th International Conference on Agents and Artificial Intelligence Volume: 1 Pages: 709–714 Year: 2014

Selbstständigkeitserklärung

Hiermit erkläre ich, Khalid Ebanhesaten, die vorliegende Dissertation selbstständig, nur unter Zuhilfenahme der aufgeführten Quellen und Hilfsmittel, verfasst zu haben.

Grevenbroich, 01.07.2014 Khalid Ebanhesaten