Technische Universität Hamburg-Harburg
Arbeitsgebiet Technische Logistik
Prof. Dr. R. Möller
Prof. Dr.-Ing. G. Pawellek

Studienarbeit

SimTesT

Entwicklung eines Test- und Präsentationswerkzeugs für das Simulationsmodell eines Lagers

Verfasser: Felix Eckhardt
15759

Erstgutachter: Prof. Möller

Zweitgutachter: Prof. Pawellek

Betreuer: Axel Schönknecht, Sebastian Wandelt

Hamburg , 15.04.2008

Inhaltsverzeichnis

1. Einleitung 6

1.1. Vorbemerkungen 6

1.2. Ziel der Arbeit 10

1.3. Funktionalität des zu entwickelnden Tools 11

1.4. Aufbau der Arbeit 11

2. Stand der Technik 13

2.1. Lager und Lagerverwaltungssysteme 13

2.1.1. Das Lager 13

2.1.2. Lagerverwaltungssysteme 18

2.2. Datenübertragung 28

2.2.1. Datenaustausch zwischen dem Lager und dem LVS 28

2.2.2. Datenübertragung mit Safeline 29

2.2.3. Datenübertragung mit anderen Standards 32

2.3. Der Begriff Qualität im Allgemeinen 33

2.3.1. Definition: Qualität 33

2.3.2. Qualität von Produkten 34

2.4. Die Software-Qualität im speziellen 37

2.5. Qualitätsmanagement und Qualitätssicherung 41

2.6. Der Softwaretest 42

2.6.1. Definition: Test 42

2.6.2. Arten von Softwaretests 43

2.6.3. Testfälle 48

2.6.4. Softwaretest und –prüfung 49

2.6.5. Der automatische Softwaretest 52

2.6.6. Unterstützung des Testens durch Tools 53

2.6.7. Grenzen von Softwaretests 56

3. Konzept zum Testen von Lagerverwaltungssystemen 57

3.1. Überblick 57

3.2. Test eines LVS mit einer Simulation 58

3.2.1. Vorgehensweise 58
3.2.2. Testarten 59
3.2.3. Grenzen 60
3.2.4. Testfälle 61
3.2.5. Protokollierung 63
3.2.6. Simulationstools 65
3.3. Weitere Möglichkeiten das LVS zu testen 66
3.4. Weitere Einsatzmöglichkeiten von Simulationsmodellen zu Testzwecken 66
3.5. Test einer Simulation 67
3.5.1. Testansätze um eine Simulation zu testen 67
3.5.2. Software zum Testen von Simulationen 68
4. Analyse 73
4.1. Use cases Test 73
4.1.1. Telegramm erstellen 74
4.1.2. Testfall erstellen 74
4.1.3. Telegramm senden 74
4.1.4. Telegramm empfangen 74
4.1.5. Testfall senden (Test starten) 74
4.1.6. Antwort empfangen 74
4.1.7. Antwort prüfen (Antwort mit DB vergleichen) 75
4.2. Use Cases Datengenerierung 75
4.2.1. Aviseposition anlegen 76
4.2.2. Avise anzeigen 76
4.2.3. Aviseposition löschen 76
4.2.4. Aviseposition bearbeiten 76
4.2.5. Avise Liste importieren 76
4.2.6. Avise vollständig löschen 76
5. Entwürfe 77
5.1. Übersicht 77
5.2. Entwurf für eine Klasse Testtool 77
5.3. Entwurf für die Telegramm Objekte 77

5.4. Entwurf für ein Objekt Testfall 78
5.5. Entwürfe für ein LVS Interface zur Testfallgenerierung 79
5.5.1. Zu erzeugende Telegramme (senden) 79
5.5.2. Zu erzeugende Telegramme (empfangen) 79
5.5.3. Benötigte Daten 79
5.6. Aktivitätsdiagramme 80
5.7. Datenbankentwurf 82
6. Eingesetzte Software 83
6.1. Ausgangssituation 83
6.2. Programmiersprache 83
6.3. Webserver und Datenbank 86
6.4. Templates zur Darstellung des Administrationsinterfaces 86
6.5. Entwicklungsumgebung 88
6.6. Softwareübersicht und Versionen 88
7. Fazit 89
7.1. Umgesetzte Funktionalität 89
7.2. Persönliches Fazit zur eingesetzten Software 94
Literaturverzeichnis 96
Abkürzungsverzeichnis 101
Tabellenverzeichnis 103
Abbildungsverzeichnis 104
Code Verzeichnis 106
Code Formatierung 107
Danksagung 109

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig erstellt, keine anderen als die angegebenen Literaturstellen benutzt und keine fremde Hilfe in Anspruch genommen habe.

Unterschrift des Verfassers

Hamburg, den

1. Einleitung

1.1. Vorbemerkungen

Läger sind komplexe logistische Anlagen [ARN-02, S. C2-50], die große Mengen Waren und Güter ein- und auslagern und in kürzester Zeit auftragsgerecht zur Verfügung stellen müssen. Häufig werden Läger auch als Logistikzentren bezeichnet. Beispielhaft seien das größte Hochregallager Europas in Bremen (betrieben von Tchibo), das Otto Versandhaus in Haldersleben und das Logistikzentrum der Kühne + Nagel International AG in Altenwerder (sieheAbbildung 1.1) genannt.

Abbildung 1.1 *Logistikzentrum der Kühne + Nagel International AG (Quelle Wikimedia Commons)*

Diese komplexen Anlagen sind ohne Softwareunterstützung nicht mehr steuerbar [GLE-08 S. 216]. Die dafür eingesetzten Softwaresysteme werden in der Regel als Lagerverwaltungssysteme (LVS) bezeichnet. „Sie zählen zu den komplexesten Software-

Systemen, die wir überhaupt in der Industrie haben“, wird Professor Michael ten Hompel, Leiter des Fraunhofer Institutes für Materialfluss und Logistik auf [CHA-08] zitiert. Als Ursache wird von ihm die extreme Flexibilität angeführt, die von einem Lagerverwaltungssystem gefordert wird. Aufgrund der unterschiedlichen Anforderungen und Realisierungsvarianten sind die Möglichkeiten der Standardisierung relativ klein. Stattdessen sind offene, modularisierte Anwendungen gefordert. Trotz der Modularisierung sind jedoch häufig Softwareanpassungen aufgrund von Kundenwünschen notwendig. [ARN-02, S. C2-50]

Wird ein Softwaresystem angepasst oder verändert existiert ein Fehlerrisiko. Nicht umsonst gibt es das Sprichwort:

If it ain't broken, don't fix it![1]

[LUD-07 S.540]

Führt ein Fehler zu einem Ausfall des Lagerverwaltungssystems oder zu einer verspäteten Inbetriebnahme eines Logistikzentrums, entstehen dabei enorme Kosten. Der Gesamte Betrieb muss eingestellt werden bis das LVS wieder ordnungsgemäß funktioniert. Ein Lagerverwaltungssystem muss also hohen Qualitätsansprüchen gerecht werden.

Neben einer sorgfältigen Entwicklung des Lagerverwaltungssystems helfen umfangreiche Testverfahren die Qualität des Softwaresystems zu erhöhen. Das Dilemma besteht jedoch darin, dass im Regelfall die Software erst mit der realen Logistikanlage getestet werden kann. Diese ist aber zumeist ebenfalls erst mit Fertigstellungsdatum der Software vorhanden und gegebenenfalls auch nicht fehlerfrei. Logistische Simulationen, das sind spezialisierte Programme zum Abbilden von Fördertechniken, Gabelstaplern und anderen Logistischen Elementen können hier im Vorfeld Unterstützung schaffen.

[1] Wenn es nicht kaputt ist, repariere es nicht!

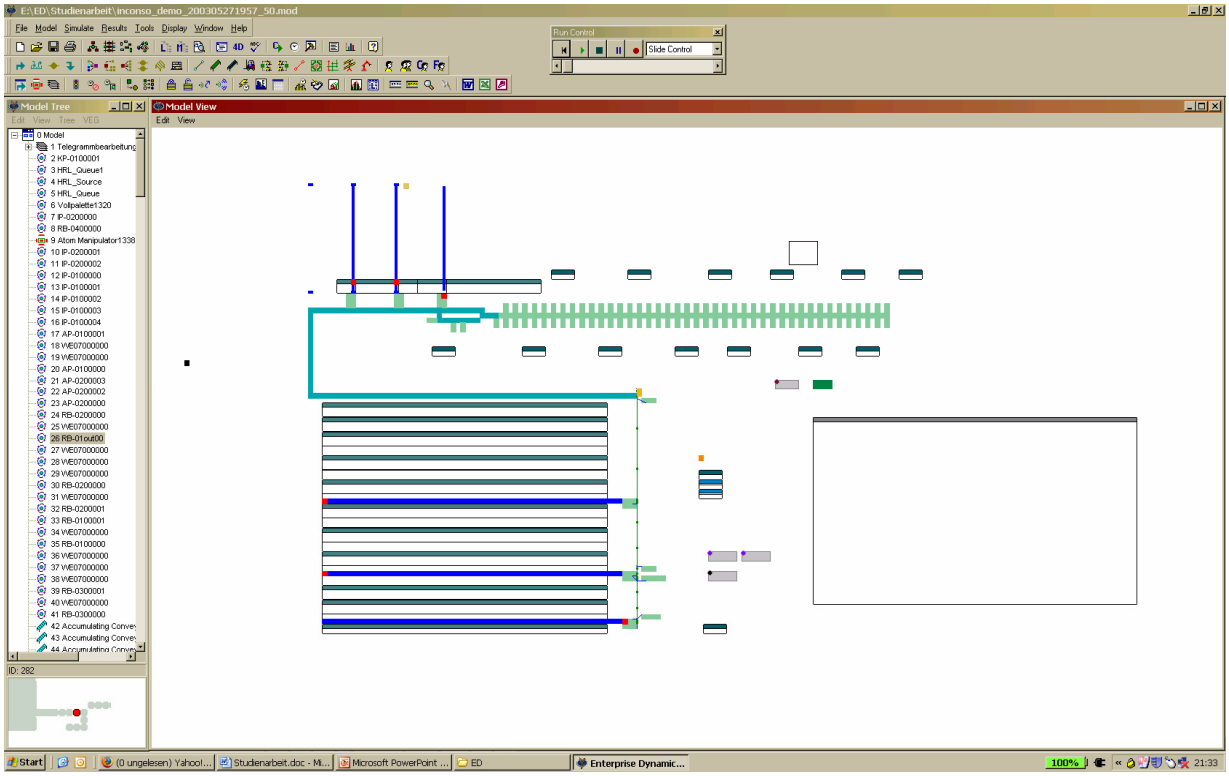


Abbildung 1.2 *Simulationstool Enterprise Dynamics (2D Modell, Screenshot)*

Mit einer Simulation (Beispiele für Simulationstools sind in Abbildung 1.2 und Abbildung 3.3 gezeigt) kann man ein LVS Testen, indem man das Verhalten des realen Lagers in einem Modell abbildet und das LVS die Simulation steuern lässt. So ist es möglich Fehler im Lager z.B. einen defekten Sensor nachzuahmen und festzustellen, wie die Software damit umgeht. Auch können verschiedene Module bzw. Anwendungsfälle des LVS (z.B. die Inventurzählung) im Vorfeld getestet werden, die erst weit nach der Inbetriebnahme des Lagers real auftreten. Die Qualität des Tests hängt maßgebend von der Qualität des Testgeschirrs[2] (siehe Kapitel 2.6.4.2) ab. Dabei ist es nicht zwingend nötig das Lager exakt nachzubilden. Eine logische bzw. qualitative Abbildung der Lagerabläufe ist ausreichend [PAW-07, S.10].

Man kann also Fehler im LVS nachweisen und diese vor Beginn der Einführung beheben. Der Fehler würde also keine Folgen in der Realität haben [PAW-04]. So verkürzt man zusätzlich die Einführungszeit und senkt Risiko und Termindruck [SPI-06 S.10]. Die

[2] in diesem Fall also von der Simulation

Einführungszeit ist extrem kostenintensiv, da zum Testen der Software; wie bereits erwähnt, das gesamte Lager inklusive Personal zur Verfügung stehen muss.

Ein LVS hat neben der Steuerung des Lagers weitere Funktionen (z.B. die Kommunikation mit ERP Systemen). Diese müssen selbstverständlich auch getestet werden.

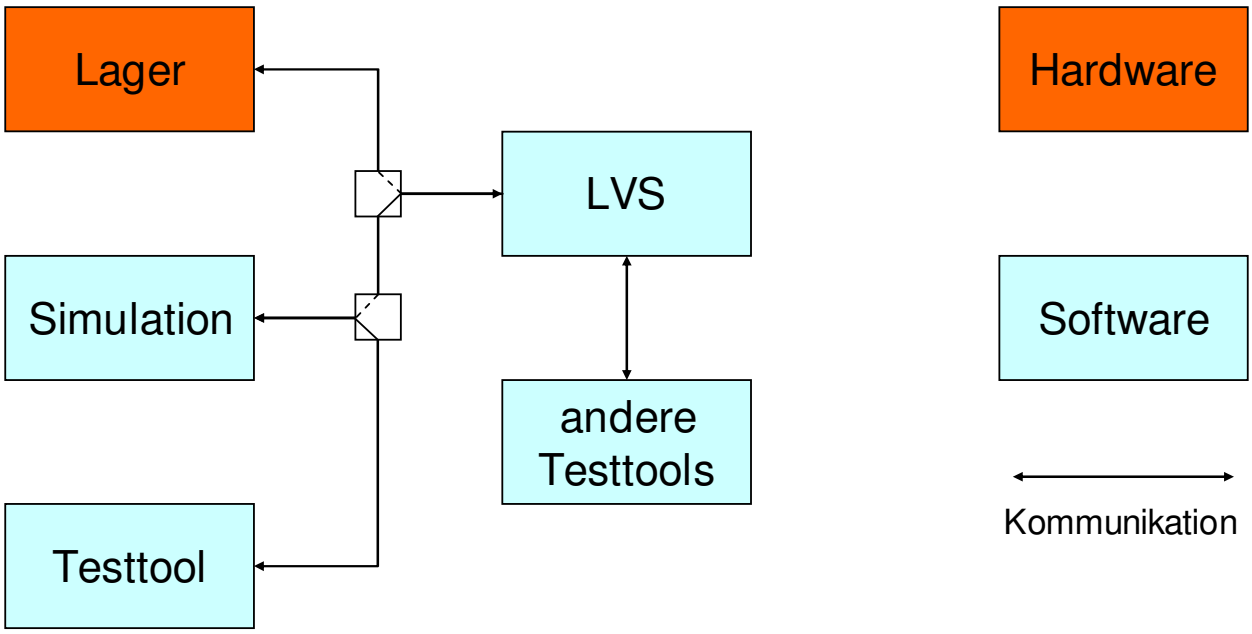


Abbildung 1.3 *Testsituation (eigene Darstellung)*

Die Simulation ist aber ebenfalls eine komplexe Software und damit fehleranfällig. Auch sie muss getestet werden, da sonst der Aufbau des eigentlichen Tests (siehe Abbildung 1.3) zwischen echtem LVS und der Simulation sehr zeitaufwändig wird.

Die Installation eines realen Lagerverwaltungssystems ist aus lizenzrechtlichen Gründen problematisch (es sei denn die Entwicklung der Simulation geschieht durch den Hersteller des LVS) und zudem sehr teuer. Zusätzlich müssen Lagerverwaltungssysteme an jedes Lager angepasst werden und stehen während der Entwicklung der Simulation nicht unbedingt zur Verfügung.

Es ergibt sich daraus eine zeitliche Reihenfolge der Testabläufe, die in dargestellt Abbildung 1.4 ist. Die Simulation wird durch das zu erstellende Testtool getestet. Das LVS dagegen wird getestet, indem es mit der Simulation statt dem Lager kommuniziert. Zusätzlich kann

der Test des LVS auch durch andere Tools unterstützt werden. Zum Einsatz können z.B. sogenannte capture and replay Tools kommen (siehe dazu auch Kapitel 2.6.6).

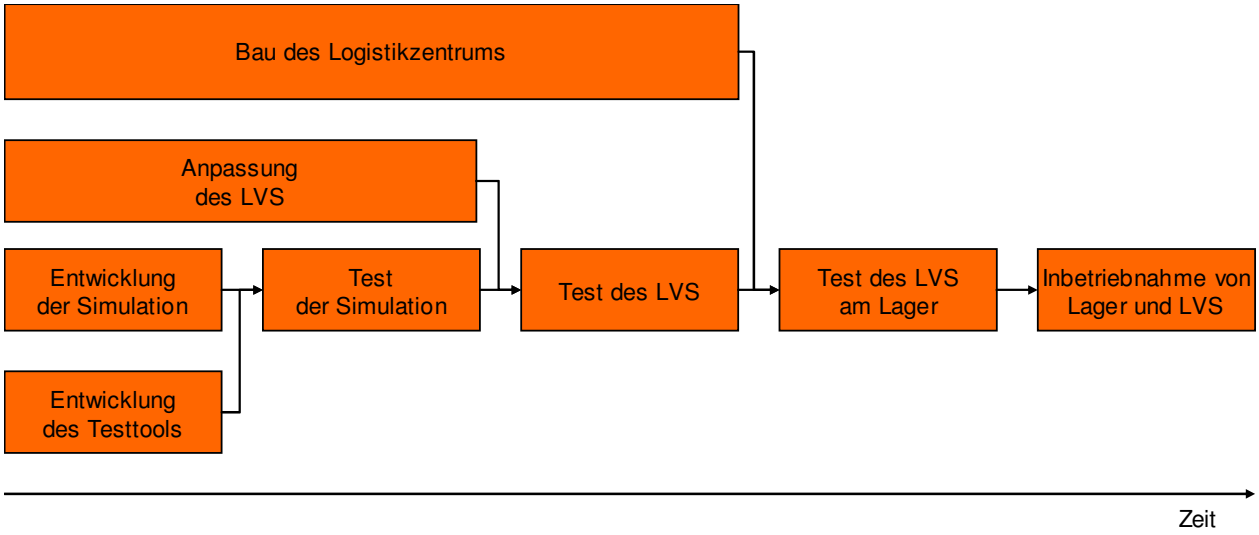


Abbildung 1.4 *Testreihenfolge (eigene Darstellung)*

1.2. Ziel der Arbeit

Das Ziel der Arbeit gliedert sich in die permanenten Bemühungen ein, Softwaresysteme qualitativ hochwertig und fehlerfrei bereits bei Übergabe auszuliefern. Im Rahmen dieser Gesamtzielsetzung soll ein Test-Lagerverwaltungssystem mit ausgewählten Funktionen entwickelt werde, dass eine komplexe Lagersimulation bedingt steuern kann. Die komplexe Lagersimulation ist ihrerseits die Softwaretestumgebung, um später das reale Lagerverwaltungssystem zu testen

Sendet man fehlerhafte Nachrichten (Telegramme) an die Simulation, sieht man die Auswirkungen im Simulationsmodell. Paletten können z.B. an einen definierten Ort gefahren werden, da z.B. die Einlagerziele nicht bekannt sind oder falsch übermittelt wurden. In realen Logistikanlagen existiert dafür oft ein „Nicht in Ordnung“ oder NiO Ort (siehe auch Abbildungen Abbildung 3.1 und Abbildung 3.2). Dieses Verhalten zeigt später dem LVS-Entwickler, dass ein (logisches) Fehlverhalten in seiner Software vorliegt. Dieses Fehlerverhalten muss aber auch die Simulation erst "erlernen". Deshalb benötigt sie ihrerseits ein Prüf-LVS.

Ein solches Tool stellt ein rudimentäres LVS dar, daher ist es von Seiten des Arbeitsgebiets Technische Logistik gewünscht das Tool über ein Webinterface zu Bedienen. Dies ist in Lagerverwaltungssystemen heute weit verbreitet.

Die Arbeit wird außerdem darstellen, welche Möglichkeiten es gibt Lagerverwaltungssysteme und Simulationen zu testen. Dazu sollen die Themen Qualität und Testen in diesem Zusammenhang erläutert werden. Da ein LVS ein Lager steuert und eine Simulation dieses abbildet, wird zu Beginn der Arbeit auch kurz auf Lagerstruktur und Abläufe in Lägern eingegangen.

1.3. Funktionalität des zu entwickelnden Tools

Die zu implementierende Software soll in der Lage sein, eine einfache Simulation zu steuern. Dazu werden Telegramme[3] an die Simulation über TCP/IP (OSI - Schichten 3 und 4) übertragen. Des Weiteren sollen Antworten automatisch geprüft und ein Protokoll erstellt werden. Die Kommunikation soll über das in [BLG-02b] beschriebene Safeline Standard abgewickelt werden. Eine Beispielsimulation wird durch das Arbeitsgebiet „Technische Logistik“ zur Verfügung gestellt. Die Simulation modelliert das im letzten Abschnitt erwähnte Vollpalettenlager der Firma Tchibo. Das Simulationstool, in dem dieses realisiert ist heißt Enterprise Dynamics. Informationen zu diesem Tool sind [ENT-08] zu entnehmen.

1.4. Aufbau der Arbeit

Diese Arbeit geht zunächst auf die Struktur von Lägern ein und macht deutlich, was ein Lagerverwaltungssystem (LVS) leisten muss, um ein Lager zu steuern. Anschließend werden folgende Themen behandelt:

- Der Begriff Qualität im Allgemeinen
- Softwarequalität im speziellen
- Qualitätsmanagement
- Der Test
- Softwaretest und –prüfung, sowie deren Grenzen
- Automatisches Testen

[3] eine Definition dieses Begriffs befindet sich in Kapitel 2.1.1.3

- Testen von Software durch Tools

Darauf aufbauend wird ein Konzept zum Testen von Lagerverwaltungssystemen vorgestellt. Das Testen von Simulationen wird im Rahmen dieses Konzeptes beleuchtet. Ein solcher Test kann durch ein Tool unterstützt werden. Daher geht die Arbeit auch darauf ein, was ein solches Tool beherrschen muss.

Im Anhang befinden sich Angaben zur Analyse und zum Entwurf des Testtools, sowie die technische Dokumentation und Angaben zur eingesetzten Software.

2. Stand der Technik

2.1. Lager und Lagerverwaltungssysteme

2.1.1. Das Lager

2.1.1.1. Definition: Lager

Dem Logistik Lexikon [LOG-08] ist folgende Definition zu entnehmen:

Ort der Vorratshaltung mit entsprechenden Einrichtungen zur Aufnahme, Kommissionierung und Spedition von Material

Eine weitere Definition liefert das Taschenlexikon Lexikon [HOM-08 S. 156]:

Sind Räume oder Flächen zum Aufbewahren von Materialien und Gütern zwecks Bevorratung, Pufferns und Verteilens, sowie zum Schutz vor äußeren ungewollten Einfüssen (z.B. Witterung) und Eingriffen (z.B. unberechtigte Entnahme).

In einem Lager wird also Material angenommen, vorgehalten, zusammengestellt, wieder abgeholt und dabei vor Schäden oder Verlust geschützt. Zusätzlich sind in machen Lägern weitere Funktionen implementiert (z.B. eine Qualitätssicherung, siehe dazu auch Kapitel 2.1.1.2).

In der Literatur werden Lagerverwaltungssysteme auch als Systeme beschrieben, die notwendig sind, um ein Lager zu betreiben. Dazu gehören unter anderem auch Gabelstapler, Regalbediengeräte oder Roboter [MAR-04 S. 479 - 483]. Im Allgemeinen wird jedoch die Software, als Lagerverwaltungssystem bezeichnet.

2.1.1.2. Struktur und Abläufe

In der Literatur lassen sich unterschiedliche Definitionen der Lagerstruktur finde. Ein Lager besteht nach [KOE-03 S. 328] aus vier Zonen:

- Vorzone mit Wareneingang und Warenausgang
- Lagerzone
- Lagerbedienzone
- Kommissionierzone

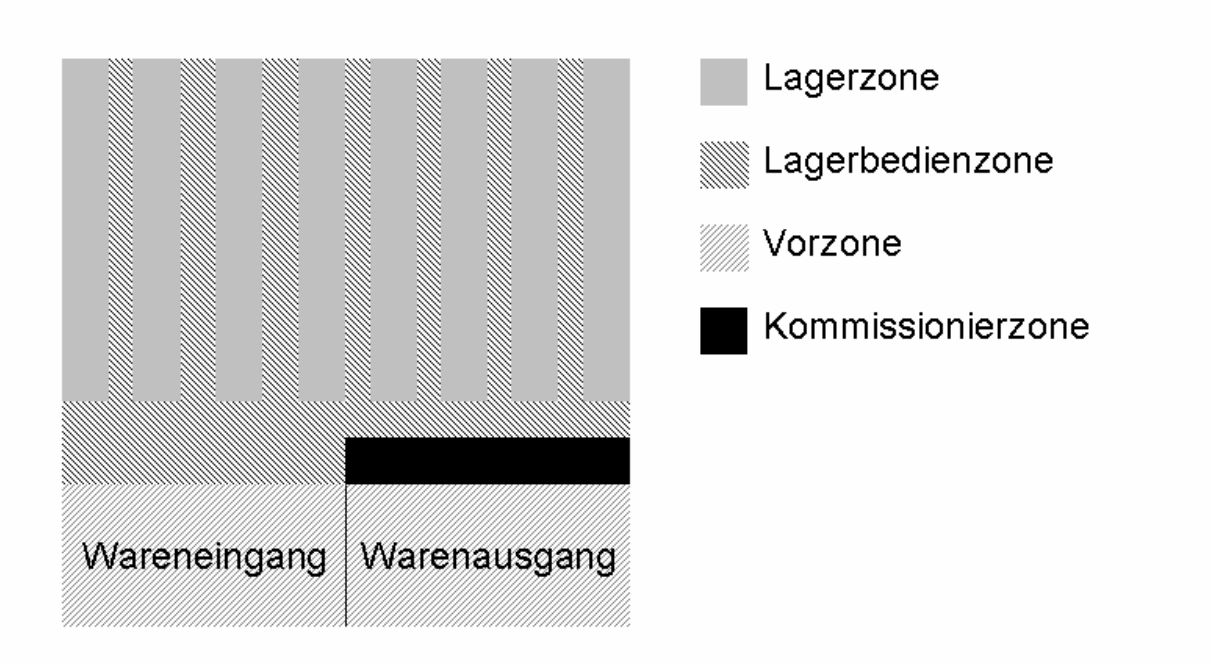


Abbildung 2.1 *Beispiel für Lagerzonen (eigene Darstellung)*

Die Abbildung 2.1 zeigt beispielhaft die Zonen in einem Lager nach dieser Definition. Die Lagerzonen stellen z.B. Regale dar, die Lagerbedienzone die Gänge auf denen z.B. Gabelstapler oder Regalbediengeräte operieren. In der Vorzone befinden sich sowohl Warenausgang als auch Wareneingang. In diesem Bereich werden alle Arbeiten durchgeführt, die bei Anlieferung und Abholung notwendig sind. In der Kommissionierzone werden die benötigten Mengen entnommen und unter Umständen auch mehrere Artikel zu einem Auftrag zusammengeführt. Das reale Layout eines Ersatzteillagers zeigt Abbildung 2.2

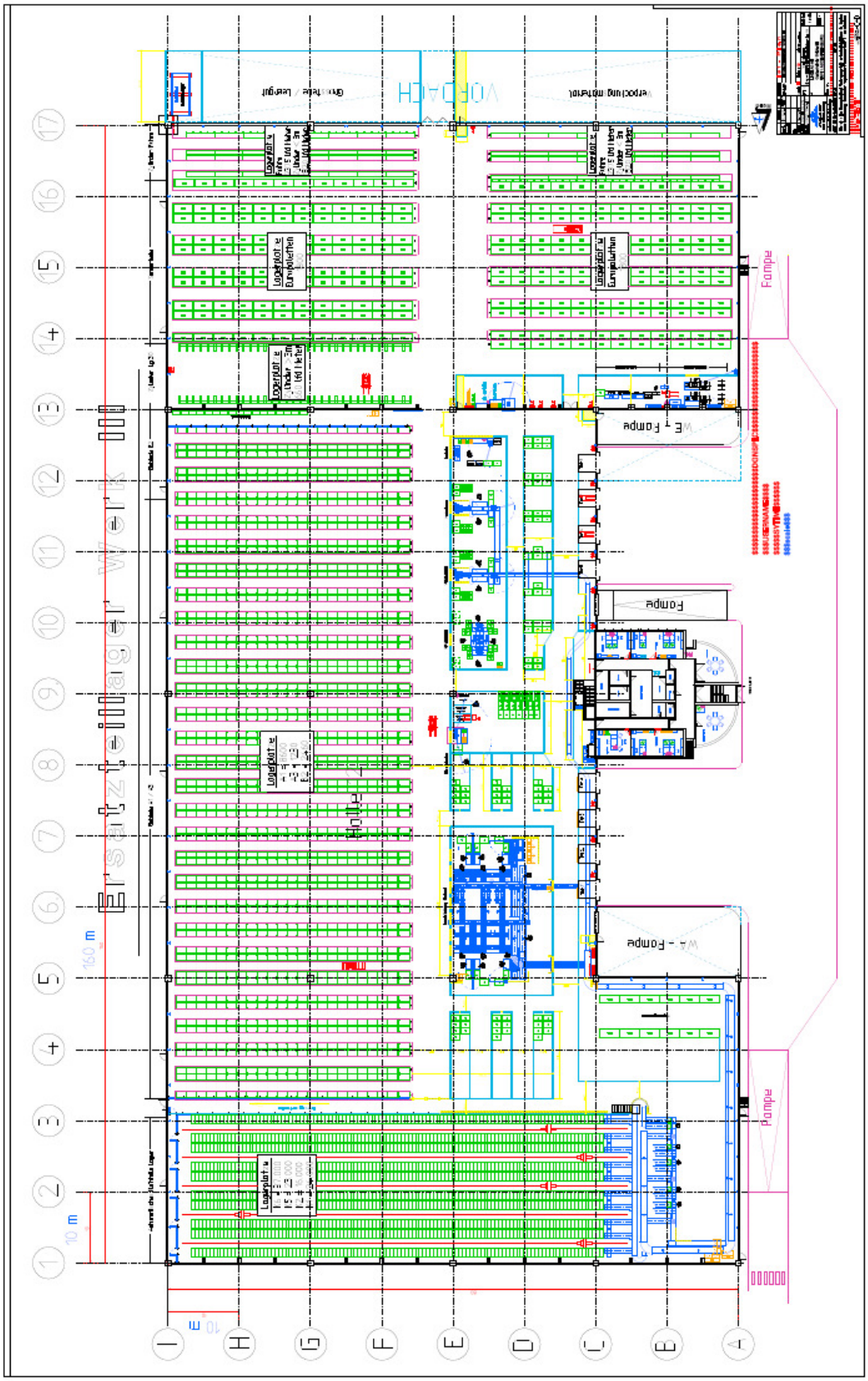


Abbildung 2.2 *Lagerlayout eines Ersatzteillagers der Kion Group (ehemals Linde, Quelle: Arbeitsgebiet Technische Logisistik)*

Eine alternative Definition der Lagerbereiche findet sich in [PFO-00 S. 130], [PAW-07a, S. 6] und in [KLA-04 S. 265]. Demnach besteht ein Lager aus fünf Bereichen (siehe auch Abbildung 2.3)

- Wareneingang
- Einheitenlager
- Kommissionierlager
- Packerei
- Warenausgang

Zusätzlich können Bereiche für die Kommissionierung oder für Veredelungsprozesse notwendig sein. Einheitenlager und Kommissionierlager können gegebenenfalls zusammengefasst sein. Die zweite Definition ist in der Literatur weit verbreitet.

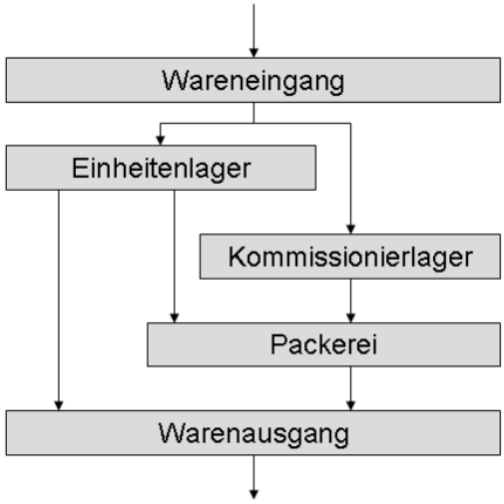


Abbildung 2.3 *Bereiche in Lägern, nach [KLA-04 S.265] (eigene Darstellung)*

Material kann abhängig von unterschiedlichen Rahmenbedingungen auf verschiedenen Wegen ein Lager durchlaufen. So kann eine bereits verpackte Palette vom Einheitenlager direkt zum Warenausgang transportiert werden. Wird jedoch nur ein einzelnes Teil benötigt, so muss dieses vorher noch in der Packerei verpackt werden.

Weitere im Lager oft vorhandene Funktionsbereiche sind:

- QS (Qualitätssicherung)
- Versandzone
- Puffer

2.1.1.3. Steuerung von Lägern

Die Steuerung von Lägern erfolgt heute mit verschiedenen Softwaresystemen auf unterschiedlichen Ebenen. In den meisten Unternehmen existiert ein ERP (Enterprise Resource Planning) System wie SAP. Es ist möglich ein Lagerverwaltungssystem in ein ERP System zu integrieren (siehe z.B. LES, dass in SAP integriert ist, Kapitel 2.1.2.5). Es gibt jedoch auch eigenständige Lagerverwaltungssysteme. Diesen können weitere, wie z.B. Materialflussrechner oder Staplerleitsysteme untergeordnet sein. Die unterste Steuerungsebene stellen z.B. SPS (Speicherprogrammierbare Steuerung), Gabelstapler oder Mitarbeiter dar. Die unterschiedlichen Ebenen sind exemplarisch in Abbildung 2.1 dargestellt. Die Anzahl der über- und untergeordneten Systeme hängen von den Anforderungen des Lagers und vom Lagertyp ab.

Der Austausch von Informationen zwischen den einzelnen Systemen geschieht über Telegramme. Ein Telegramm ist ein Datensatz mit einer standardisierten Struktur. Die Übermittlung erfolgt üblicherweise digital, jedoch existieren auch noch alte analoge Protokolle (z.B. DUST).

Ein LVS kann mit unterschiedlichen Partnern kommunizieren, um das Lager zu steuern. Neben dem MFR gibt es z.B. das Staplerleitsystem. Stellvertretend für die Ebene zwischen LVS und der Hardware wird in dieser Arbeit der MFR verwendet.

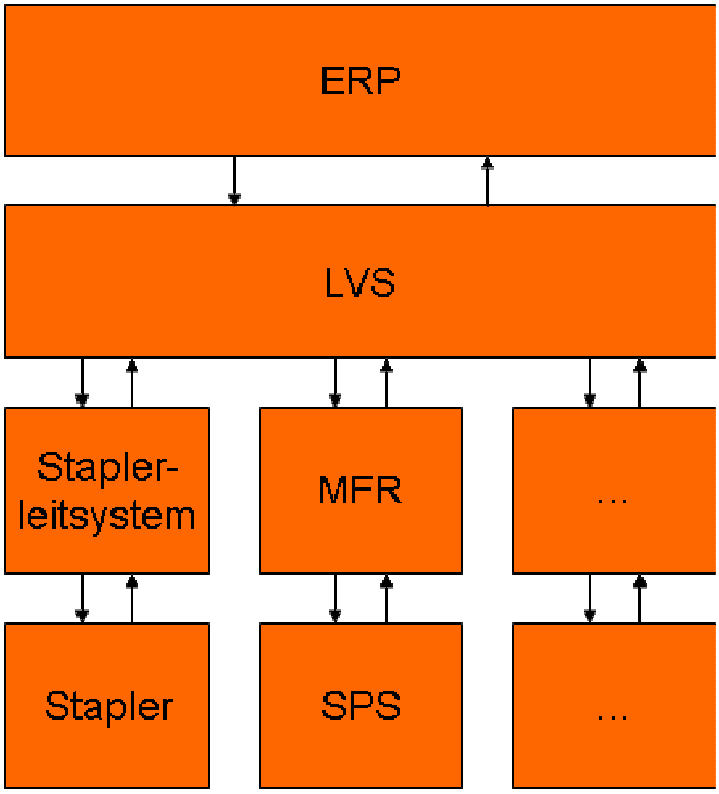


Abbildung 2.4 *Ebenen der Lagersteuerung (nach BAL-08 und GLE-08 S.217, eigene Darstellung)*

2.1.2. Lagerverwaltungssysteme

2.1.2.1. Definition: Lagerverwaltungssystem

Ein Lagerverwaltungssystem ist eine Software, die die Vorgänge in einem Lager steuert. In der Literatur lassen sich unterschiedliche Definitionen finden:

Lagerverwaltungssystem: bildet im klassischen, hierarchisch organisierten Modell die Ebene zwischen untergelagertem Materialflussrechner und überlagertem Host (ERP System)

[HOM-08]

Diese Definition ist schwammig, da Lagerverwaltungssysteme in ERP Systemen integriert sein können (z.B: LES von SAP, siehe 2.1.2.5) und auch die Funktionalität eines MFR in einem Lagerverwaltungssystem realisiert werden kann.

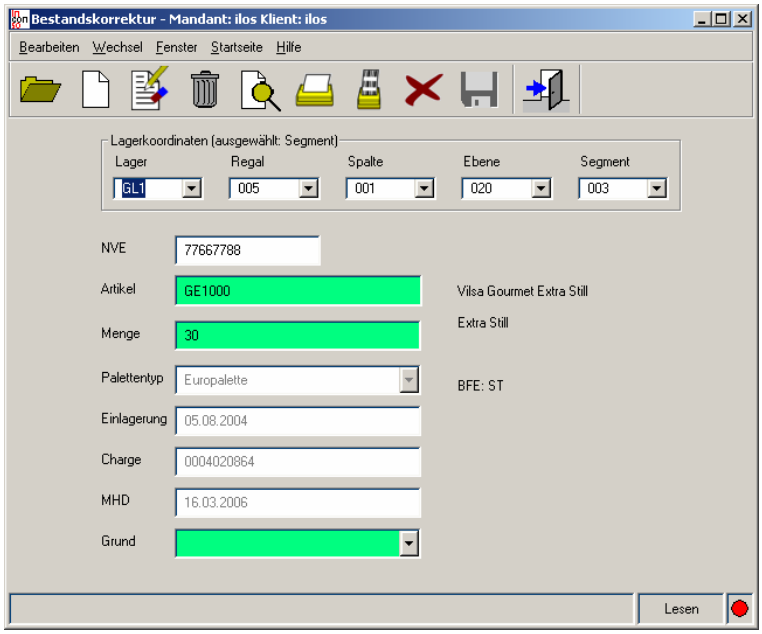


Abbildung 2.5 *Maske des incomsoWMS Standard (Bestandskorrektur) (Quelle: Programmbeschreibung inconso)*

Eine andere Definition liefert [GLE-08 S.220]:

Systeme zur übergreifenden Lagersteuerung und Bestandsverwaltung bezeichnet man als Warehouse Management Systeme (WMS) oder Lagerverwaltungssysteme (LVS)

Auch hier werden Lagerverwaltungssysteme zwischen Warenwirtschafts- bzw. ERP Systemen und Materialflussrechner eingeordnet.

2.1.2.2. Anforderungen an Lagerverwaltungssysteme

An Lagerverwaltungssysteme werden extrem gegenläufige Anforderungen gestellt. So muss sowohl eine produkttypische Stabilität als auch eine hohe Flexibilität gewährleistet sein, da einerseits Ausfälle in Lägern sehr teuer und andererseits keine zwei Läger identisch sind. Die Entwicklung der Software sollte effektiv sein, das Produkt aber dennoch eine maximale Funktionsabdeckung sowie Komplexität bieten. Zusätzlich soll ein LVS hochperformant auf unterschiedlichen Plattformen laufen [PSI-02, 2-1]. Um diese Anforderungen zu erfüllen ist es unerlässlich moderne Entwurfsmethoden und Techniken zu nutzen. Unter anderem kommt hier UML zum Einsatz. Die Plattformunabhängigkeit kann durch den Einsatz von Java (z.B. auf der Clientseite) gesichert werden. Eine weitere Möglichkeit Plattformunabhängigkeit

auf der Clientseite zu garantieren, ist der Einsatz von webbasierten User Interfaces. Dies birgt weitere Vorteile:

- Lauffähigkeit auf jeder Plattform mit Netzwerkunterstützung und Browser
- Keine Zusätzliche Software auf dem Client Rechner nötig
- Viele Menschen sind mit der Bedienung von Browsern bereits vertraut
- Daher ist die Akzeptanz dieser Technologie sehr hoch

Nachteile eines Webinterfaces:

- Prinzipiell asynchrone Kommunikation
- Darstellung unter umständen nicht in allen Browsern identisch
- Darstellung auf die Möglichkeiten beschränkt, die HTML bietet
- Unterschiedliche Interpretation von JavaScript in verschiedenen Browsern
- Permanente Verbindung zum Server nötig

Diese Nachteile lassen sich z.T. durch den Einsatz weiterer Technologien beheben. So ist die Darstellung z.B. mit Java Applets freier gestaltbar, Probleme mit JavaScript kann man so auch umgehen. Nutzt man Java, muss es allerdings auch vorhanden sein oder gegebenenfalls installiert werden.

2.1.2.3. Verwendete Standards und Techniken

Um eine maximale Flexibilität in einem LVS sicherzustellen, werden unterschiedliche Techniken und Standards verwendet. Zum Einsatz können dabei kommen:

- SQL
- CORBA
- TCP/IP
- COM/.NET
- EDI/BAPI
- XML
- IDL
- IPC
- RPC

SQL ist der allgemein gültige Industrie-Standart für Datenbank Anfragen [PSI-02, 2-1]. Die Verwendung von SQL sichert, dass unterschiedliche Datenbanksysteme eingesetzt werden können. Gebräuchliche SQL Datenbanksysteme sind z.B.:

- Oracle
- mySQL
- Microsoft SQL Server

SQL bietet die Nutzung von Transaktionen. Mit diesen kann sichergestellt werden, dass eine Reihe von Anweisungen (z.B. die Abbuchung von einem Konto und die Gutschrift auf einem anderen) entweder komplett oder gar nicht ausgeführt wird.

Neben relationalen Datenbanken kommen zum Teil auch hierarchische (z.B. das vom Hersteller als postrelationale Datenbank beschriebene Cache [INT-08]) zum Einsatz. sogar Objektorientierte Datenbanken scheinen nicht eingesetzt zu werden.

2.1.2.4. Kommunikation

Lagerverwaltungssysteme kommunizieren mit unterschiedlichen Partnern:

- dem Lager bzw. dem Materialflussrechner
- den Anwendern
- Weiterer Softwaresysteme

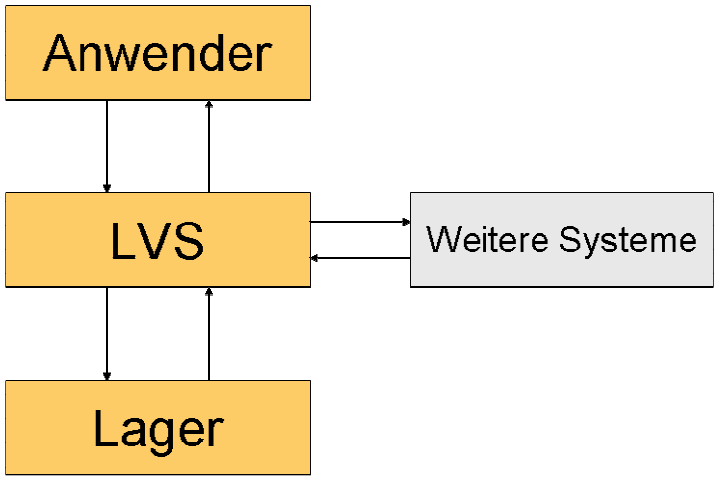


Abbildung 2.6 *Kommunikationspartner eines LVS (eigene Darstellung)*

Die Kommunikation mit dem Anwender umfasst alle Vorgänge zum Steuern des Lagers (z.B. die Auftragsfreigabe), sowie der Informationsaustausch mit den Mitarbeitern (z.B. am Kommissionierplatz). Typische Arbeitsplätze von Mitarbeitern im Wareneingang zeigt Abbildung 2.7. Weitere Systeme können z.B. ERP Systeme wie SAP oder ein Warenwirtschaftsystemen sein (siehe Kapitel 2.1.2.7). Die Kommunikation mit dem Lager umfasst z.B. Kommunikation mit Materialflussrechnern oder Staplerleitsystemen. Eine schematische Darstellung der Kommunikationspartner ist in
Abbildung 2.6 gezeigt.

Abbildung 2.7 *Buchungsarbeitsplätze im Wareneingang, Lufthansa Technik Hamburg (eigens Foto)*

2.1.2.5. Hersteller von Lagerverwaltungssystemen

Da es eine Vielzahl von Herstellern gibt beschränkt sich die folgende Tabelle auf einige im deutschsprachigen Raum vertretenen Hersteller von Lagerverwaltungssystemen.

Hersteller	**System**	**Anmerkungen**	**Quelle**
PSI logistics	PSI WMS		PSI-02
Frauenhofer	myWMS	Open Source	FRAU-08
Swisslog	WarehouseManager		SWI-08
SAP	LES	Teil von mySAP R/3	SAP-08
Inconso	HELAS		INC-08a
Inconso	ILos		INC-08b
Inconso	StoreLiner		INC-08c
Nissen & Velten	SQL-Business	Teil der ERP Lösung SQLBusiness	NIV-08
AJE Consulting	Lossy		AJE-08
ABB Automatin	SattStore		LOG-04
Aldata Retail Solutions GmbH	G.O.L.D.STOCK LSP		LOG-04
Arvato systems GmbH	WMS		LOG-04
AT Automation Systems GmbH	ATCOLT		LOG-04
Atos Origin GmbH	ICAM+ WMS		LOG-04
BEA Elektrotechnik und Automation GmbH	MOD-LFS		LOG-04
Brandt und Partner GmbH/ Fresenius Netcare GmbH	ELVICUS		LOG-04
BSS Bohnenberg GmbH	Lasys		LOG-04
CAL Consult	B.V. CALwms		LOG-04
CAL Consult BU 400	CALwms400		LOG-04
Cat Logic B.V.	WMS ‘Distri’ 4401		LOG-04
COGLAS GmbH	COGLAS Logistiksoftware		LOG-04
CONET AG	EXceed Fulfill 4000		LOG-04
Dalosy Projecten B.V.	DWS-3000		LOG-04

DCS Transport & Logistics Solutions Ltd.	DCSi.Logistics		LOG-04
Dr.Brunthaler Industrielle Informationstechnik GmbH	Storagement		LOG-04
Dr. Thomas + Partner GmbH	TP-WMS		LOG-04
ECOLOG Logistiksysteme GmbH	CI_LOG		LOG-04
Ehrhardt + Partner GmbH & Co. KG	LFS 400		LOG-04
EXE Technologies Inc.	EXceed Fulfill 4000		LOG-04
FabLog GmbH	LOGIS 2000		LOG-04
Fujitsu Services B.V	MLS warehouse management		LOG-04
GDV Kuhn mb	Probas BLVS		LOG-04
GIGATON GmbH	LogoS		LOG-04
GUS Group AG & Co. KG	Charisma		LOG-04
HARDIS International	Reflex		LOG-04
iFD Investmanagement Fabrikautomation und Design AG	LVSS		LOG-04
Interchain B.V.	Chainware iWarehouse		LOG-04
J.D. Edwards Netherlands	PeopleSoft EnterpriseOne SCM		LOG-04
JOTA GmbH	CASSIS		LOG-04
Kasto Maschinenbau GmbH & Co. KG	KASTOlvr		LOG-04
Klug GmbH	IWACS		LOG-04
KuglerConsulting GmbH	Warehouse Management System		LOG-04
LIS Logistics & Industrial Systems N.V.	Dispatcher-WMS		LOG-04
Locus Warehouse Management Systems B.V.	Locus Warehouse Management System		LOG-04
Logarithme	LM_execution		LOG-04
LOGIM Software GmbH	ALWIS		LOG-04

Tabelle 2.1 *Auflistung von Lagerverwaltungssysteme*

Das Kürzel B.V. bedeutet *besloten vennootschap* und ist die niederländische form einer GmbH.

2.1.2.6. Kernfunktionalität von Lagerverwaltungssystemen

Eine Lagerverwaltungssoftware muss in erster Linie folgende Funktionalitäten gewährleisten ([PSI-02], [CHA-8] und [WOL-07 ab S.76]:

- Ein- und Aulagerung von Artikeln bzw. Lagereinheiten
- Umlagerungen
- Einlagermanagement
- Auftragsmanagement
- Bereitstellungsmanagement
- Bestandsführung
- Belegerstellung
- Kommissioniermanagement
- Unterstützung der Mitarbeiter bei Inventuren
- Abwicklung von Warenein- und Ausgangsprozessen
- Abbildung des Lagers (Topologiemanagement)
- Verwaltung der Transporteinheiten bzw. Lagereinheiten
- Transportmanagement
- Bereitstellung von Informationen bzw. Übersichten
- Protokollierung

2.1.2.7. Erweiterte Funktionalität

Nach [PSI-02], [CHA-08a], [ARN-02] und [WOL-07 ab S. 76] zählen die folgenden zur erweiterten Funktionalität:

- Kommunikation zur übergelagerten Systemen wie z.B. WWS (Warenwirtschaftssystem), PPS (Produktionsplanung und –steuerung) oder ERP (Enterprise Resource Planing)
- Kommunikation mit Transport- oder Staplerleitsystem (sofern nicht im LVS integriert)
- Kommunikation mit untergelagerten System wie z.B. der FT (Fördertechnik, siehe auch Abbildung 2.8) oder RGBs (Regalbediengeräten)
- Wareneingang mit Bestell Avis
- QS (Qualitätssicherung)
- Chargenverwaltung

- Mandantenfähigkeit
- Inventur
- MHD Verwaltung (Mindesthaltbarkeitsdatum)
- Umlaufkommissionierung nach dem Prinzip Ware zum Mann
- Bypassfunktion (temporär)
- Cross Docking (Material vom Wareneingang direkt zum Warenausgang)
- Drucken von Etiketten
- Stücklistenverwaltung
- Dynamische Lagerplatzverwaltung
- Verdichtung
- Bewegungsjournal
- Abrufsteuerung mit zeitgesteuerter Auftragsbearbeitung
- Kommunikation mit dem Zoll
- Verwaltung von Seriennummern
- Leergutverwaltung
- Staplerleitsystem
- Dock/Yardmanagement
- Retourenmanagement
- Doppelt/Mehrfachtiefe Lagerung
- VAS (Value addede Services, Kundenservice)
- VMI (Vendor Managed Inventory, Lieferanten gesteuerter Bestand)
- Gefahrstoffverwaltung
- Unterstützung von Sendungsverfolgung (Übermittlung von Paketnummern an Spediteure)
- Unterstützung von Pick und Pack Vorgängen (Verpackung am Kommissionierplatz)
- Lagervisulalisierung

Diese werden üblicherweise in Modulen implementiert, die optional beim Kunden mitinstalliert werden. Der Kunde ist so nicht mehr gezwungen das ganze System zu erwerben, sondern kann einzelne Module (auch im Nachhinein) dazukaufen. [ARN-07 S.76]

Abbildung 2.8 *Fördertechnik bei der Lufthansa Technik Hamburg (eigenes Foto)*

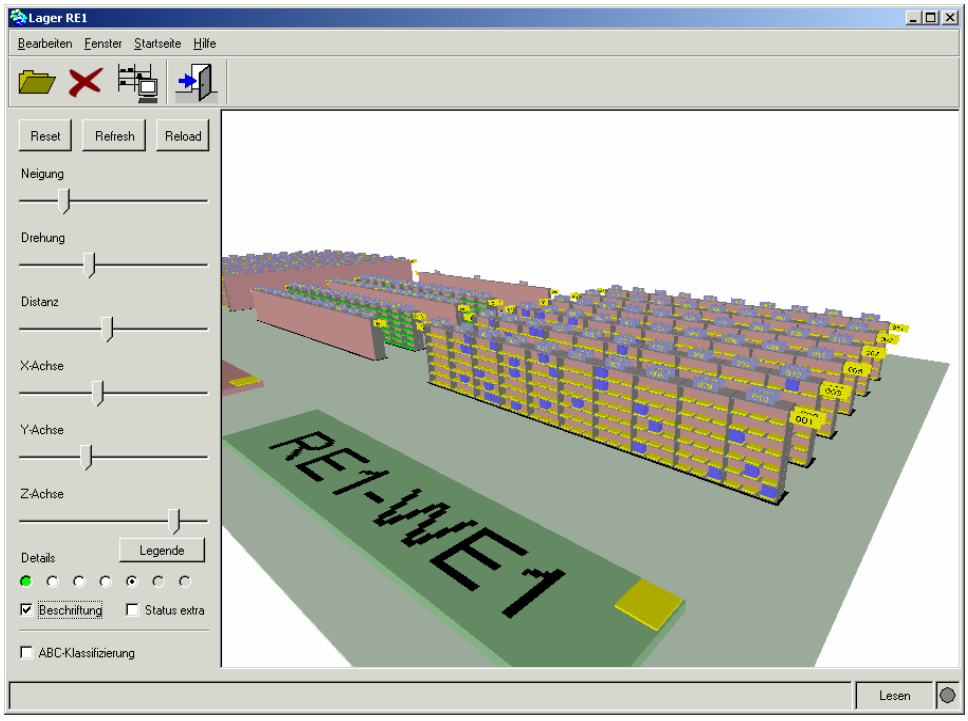


Abbildung 2.9 *Lagervisualisierung im inconso StandardWMS (Quelle: Programmbeschreibung)*

2.1.2.8. Betriebsarten von Lagerverwaltungssystemen

Nach [MAR-04 S. 475 – 483] werden Lagerverwaltungssysteme in drei unterschiedlichen Betriebsarten eingesetzt:

1. Offline Betrieb: diskontinuierliche Datenübertragung zwischen Zentralrechner und Peripheriegeräten (auch indirekte Kopplung genannt)
2. Online Betrieb: kontinuierliche Datenübertragung zwischen Zentralrechner und Peripheriegerät über eine feste Verbindung (auch geschlossene Kopplung oder closed loop genannt)
3. Direkte offene Kopplung: Von einer direkten offenen Kopplung spricht man, wenn entweder die Eingabe- oder Ausgabeseite von einem Menschen bedient wird.

2.2. Datenübertragung

2.2.1. Datenaustausch zwischen dem Lager und dem LVS

Damit das LVS das Lager steuern kann, müssen Informationen und Anweisungen ausgetauscht werden. So muss der MFR dem LVS Beispielsweise die Information zukommen lassen, dass eine Palette an einem bestimmten Punkt im Lager angekommen ist. Als Reaktion darauf wird das LVS z.B. eine Anweisung zur Einlagerung an den MFR senden.

Die Übertragung der Telegramme erfolgt im Allgemeinen unverschlüsselt, da Punkt zu Punkt Verbindungen bestehen. Bei der Übertragung von Informationen über das Internet sind entsprechende Sicherheitsmaßnahmen zu ergreifen.

Bei dem Test eines LVS mit einer Simulation sendet die Simulation die gleichen Telegramme, wie der MFR. Das LVS kann dann zwischen MFR und Simulation nicht unterscheiden.

2.2.2. Datenübertragung mit Safeline

2.2.2.1. Übertragung der Telegramme

Alle Angabe zu Safeline sind [BLG-02b] entnommen.

Die Übertragung von Informationen geschieht im Safeline Standard über TCP Verbindungen. Über diese Verbindungen werden Telegramme zwischen dem LVS und dem Materialflussrechner (MFR) ausgetauscht. Insgesamt werden zwei Verbindungen aufgebaut. Über die erste Verbindung werden Telegramme vom LVS an den MFR gesendet und Empfangsquittungen vom MFR empfangen. Über die andere Verbindung werden Telegramme vom MFR an das LVS gesendet. Auch das LVS quittiert den Empfang jedes Telegramms. Die verwendeten Ports am LVS sind 8101 und 8102, Lagerseitig verwendet der MFR unterschiedliche Ports. Grundsätzlich stellt der Materialflussrechner die Verbindung zum LVS her. Dieses startet also einen Dienst, der auf Verbindungsanfragen wartet.

Lagerverwaltungssysteme müssen Daten mit einem MFR und einer Client Software austauschen. Die Clientsoftware teilt z.B. einem Kommissionierer mit wie viele Teile benötigt werden. Unter Umständen muss weitere Kommunikation z.B. mit einem Staplerleitsystem erfolgen.
Das in Kapitel 1.1 bereits angeführte Vollpalettenlager von Tchibo arbeitet mit diesem Standard. Dieser Arbeit liegt eine Simulation zu Grunde, die das Tchibo Lager abbildet.

2.2.2.2. Telegramme im Safeline Standard

Der Datenaustausch zwischen dem Lagerverwaltungssystem und dem MFR findet mit Telegrammen statt. Der Standard definiert 14 unterschiedliche Telegrammarten.

Telegrammtyp	Von	An	Bemerkung
01	LVS	MFR	Auslagerauftrag
02	MFR	LVS	Ankunftsmeldung
03	MFR	LVS	Platzfreimeldung
04	LVS	MFR	Storno Auftrag (Stornierung eines Auftrags)
05	LVS	MFR	Folgeauftrag
06	MFR	LVS	Auftragsabbruch

10	MFR	LVS	Anlagenstatusmeldung
80	LVS	MFR	An-/Abmeldung Leerpalettensendung
81	MFR	LVS	Rückmeldung für Leerpalettensendung (Antwort auf 80)
82	LVS	MFR	Artikelan-/abmeldung WE lose
83	LVS	MFR	Freigabe Vortakten WE-Bahn
84	LVS	MFR	Freigabe Einlagerung WE-Bahn
85	LVS	MFR	Freigabe Verladung WA-Bahn
86	MFR	LVS	Anforderung Nachschub

Tabelle 2.2 *Telegramme im Safeline Standard*

Zusätzlich zu den in Tabelle 2.2 aufgeführten Telegrammen kommt im Safeline Standard noch das Quittungstelegramm zum Einsatz, das den Empfang eines Telegramms bestätigt.

Die Telegrammarten 01 bis 06 sind identisch aufgebaut. Damit ergeben sich neun unterschiedliche Telegrammarten. Die Datentypen der einzelnen Felder werden als C (Charakter / Zeichen) oder N (Numerisch) angegeben.

2.2.2.3. Aufbau der Telegramme: LVS -> MFR

Folgende Telegramme werden vom LVS an den MFR geschickt:

Telegrammtyp	**Von**	**An**	**Bemerkung**
01	LVS	MFR	Auslagerauftrag
04	LVS	MFR	Storno Auftrag (Stornierung eines Auftrags)
05	LVS	MFR	Folgeauftrag
80	LVS	MFR	An-/Abmeldung Leerpalettensendung
82	LVS	MFR	Artikelan-/abmeldung WE lose
83	LVS	MFR	Freigabe Vortakten WE-Bahn
84	LVS	MFR	Freigabe Einlagerung WE-Bahn
85	LVS	MFR	Freigabe Verladung WA-Bahn

Tabelle 2.3 *Telegramme im Safeline Standardl (LVS -> MFR)*

Beispielhaft ist der Aufbau der Telegramme 01, 04 und 05 in Tabelle 2.4 dargestellt. Ihre Struktur ist identisch.

Name	**Typ**	**Bemerkung**
Telegrammart	C(2)	01 – 06
Herkunft	C(10)	
Standort	C(10)	
Endziel	C(10)	
Status	C(2)	
LE-Nummer	C(11)	
Vorgangstyp	C(6)	
Vorgangsnummer	C(25)	
Höhenklasse	C(2)	„01“, 02“, „03“
LE-Typ	C(4)	„EP “
Gewichtsfehler	C(1)	„0“ : i.O. „G“ : Übergewicht
Gewicht Waage	N(7)	Gewogenes Gewicht von Waage vor NiO Platz (in Gramm)
Konturenfehler	C(32)	Bitstring (je Fehler ein Zeichen 0/1)
Kreiselfehler	C(2)	„00“ : i.O., „KS“ : Kreiseln wegen Störung Abgabeplatz „KB“ : Kreiseln wegen Abgabeplatz belegt
Etikettdaten	C(150)	Mehrere Daten in einem Feld (definierte Reihenfolge, durch @ getrennt)
Etikett Drucken	C(1)	J(a) bzw. N(ein)
Stretchprogramm	C(2)	
Lesefehlererkennung	C(1)	„0“ : Lesefehler nicht erlaubt, „1“ : Lesefehler erlaubt
Stapleraufnahme	C(1)	„Q“ ; Queraufnahme „L“ : Längsaufnahme
Menge	N(7)	Menge auf der LE
Priorität	N(2)	
Fehler LVS	C(2)	Fehlerstatus auf LVS
Unterpalettierung	C(1)	„0“ : keine Unterpalettierung „1“ : Unterpalettierung durchführen

Tabelle 2.4 *Telegramme 01, 04, 05 im Safeline Standard (LVS -> MFR)*

2.2.2.4. Telegramme: MFR -> LVS

Folgende Telegramme werden vom MFR an das LVS geschickt:

Telegrammtyp	Von	An	Bemerkung
02	MFR	LVS	Ankunftsmeldung
03	MFR	LVS	Platzfreimeldung
06	MFR	LVS	Auftragsabbruch
10	MFR	LVS	Anlagenstatusmeldung
81	MFR	LVS	Rückmeldung für Leerpalettensendung (Antwort auf 80)
86	MFR	LVS	Anforderung Nachschub

Tabelle 2.5 *Telegramme im Safeline Standrds (MFR -> LVS)*

2.2.3. Datenübertragung mit anderen Standards

Es existieren neben Safeline auch andere Standards zur Übermittlung von Informationen zwischen dem LVS und dem MFR. SAP setzt Beispielsweise den IDOC Standard zur Übertragung von Informationen ein [SAP-08a]. Auch XML kommt zum Einsatz.

Nachrichten können natürlich auch über eine höhere Protokollebene ausgetauscht werden.

Im EDI (Electronic Data Interchange) verbreitete Protokolle:

- X400 (E-Mail-Standard der ITU)
- FTAM (File Transfer Access and Management)
- OFTP (Odette File Transfer

Zur Datenübertragung werden z.T. auch die Internet Protokolle genutzt:

- SMTP (Simple Mail Transfer Protocoll)
- FTP (File Transfer Protocoll)
- HTTP (Hypertext Transfer Protocoll)
- AS 1-3 (Application Statement 1-3, SMTP, FTP und HTTP verschlüsselt mit der Möglichkeit auch die Leitung selber zu verschlüsseln)

Es existieren noch diverse andere Protokolle, von denen viele auf [IPC-08] aufgeführt und beschrieben sind.

2.3. Der Begriff Qualität im Allgemeinen

2.3.1. Definition: Qualität

Das Wort Qualität stammt ursprünglich von dem Lateinischen Wort qualitas. Es kann unterschiedliche Bedeutungen haben:

1. Art, Beschaffenheit
2. Eigenschaft, Fähigkeit
3. Vokalfärbung
4. Sorte, Güte

Heute wird Qualität vor allem im Sinne von guten Eigenschaften benutzt [LUD-07 S.63], [WAH-04].

Der Test ist eine Möglichkeit die Qualität zu prüfen und zu verbessern. Findet man in einem Test einen Fehler, kann man diesen beheben und so die Eigenschaften eines Produktes verbessern. Man weiß allerdings nicht, wie viele Fehler in einer Software sind. Erfahrungen haben gezeigt, dass man im Rahmen von hochwertigen Tests etwa die hälfte der Fehler in einer Software finden kann. Will man also in einer Software maximal *n* Fehler nach einer Prüfung tolerieren, dürfen vor dem Testen nicht mehr als *2n* Fehler existieren (siehe Abbildung 2.10). Testen allein kann also keine fehlerfreie (besser fehlerarme) Software sicherstellen. Eine sorgfältige Entwicklung ist dazu unumgänglich [LUD-07 S.65].

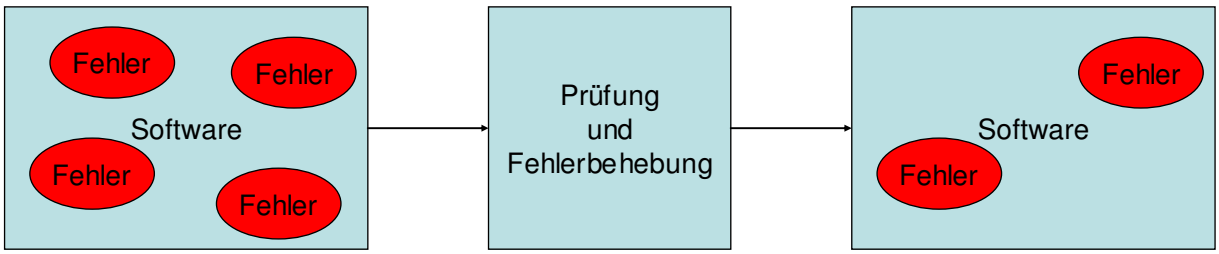


Abbildung 2.10 *Anzahl der Fehler vor und nach einer Prüfung/Fehlerbehebung (eigene Darstellung)*

2.3.2. Qualität von Produkten

In der Literatur finden sich unterschiedliche Definitionen für den Begriff Qualität. In [BAL-98 Seite 256] werden fünf Ansätze genannt:

- Der transzendente Ansatz
- Der produktbezogene Ansatz
- Der benutzerbezogene Ansatz
- Der prozessbezogene Ansatz
- Der Kosten/Nutzen bezogene Ansatz

2.3.2.1. Der transzendente Ansatz

Dieser Ansatz geht von einer universellen, absoluten, einzigartigen und erkennbaren Qualität aus, die nicht messbar und auch nicht exakt definierbar ist. Demnach lässt sich Qualität nur durch Erfahrung bewerten. Dieser Ansatz wird daher als ungeeignet für die Messung und Beurteilung von Qualität eingestuft.

2.3.2.2. Der produktbezogene Ansatz

Qualität wird in diesem Ansatz als eine messbare Größe betrachtet. Subjektive Beobachtungen und Wahrnehmungen spielen in dieser Betrachtung keine Rolle. Es werden ausschließlich die Merkmale des Produktes heran gezogen, um die Qualität zu messen. Dies kann dazu führen, dass Kundenwünsche nicht befriedigt werden.

2.3.2.3. Der benutzerbezogene Ansatz

Im benutzerbezogenen Ansatz definiert der Kunde die Qualität. Kunden haben oft nur eine vage Vorstellung davon, was sie unter Qualität verstehen. Daher kann dieser Ansatz nur bedingt dazu benutzt werden, Qualität zu messen.

2.3.2.4. Der prozessbezogene Ansatz

In diesem Ansatz wird davon Ausgegangen, dass die Qualität eines Produktes durch die Qualität des Produktionsprozesses bestimmt wird.

2.3.2.5. Der Kosten-Nutzen bezogene Ansatz

Qualität ist als Funktion von Kosten und Nutzen gegeben. Ein Qualitätsprodukt liefert also einen Nutzen zu einem akzeptablen Preis.

2.3.2.6. Der Qualitätsbegriff nach DIN und ISO Normen

Balzert folgt im Weiteren der DIN 55 350[4], die Qualität (im Gegensatz zur umgangssprachlichen Bedeutung) wertfrei definiert:

> *Qualität: Gesamtheit von Eigenschaften und Merkmalen eines Produktes oder einer Tätigkeit, die sich auf die Eignung zur Erfüllung gegebener Erfordernisse beziehen.*

[4] Genauer ist dies ein Teil der DIN 55 350-11:1995-08

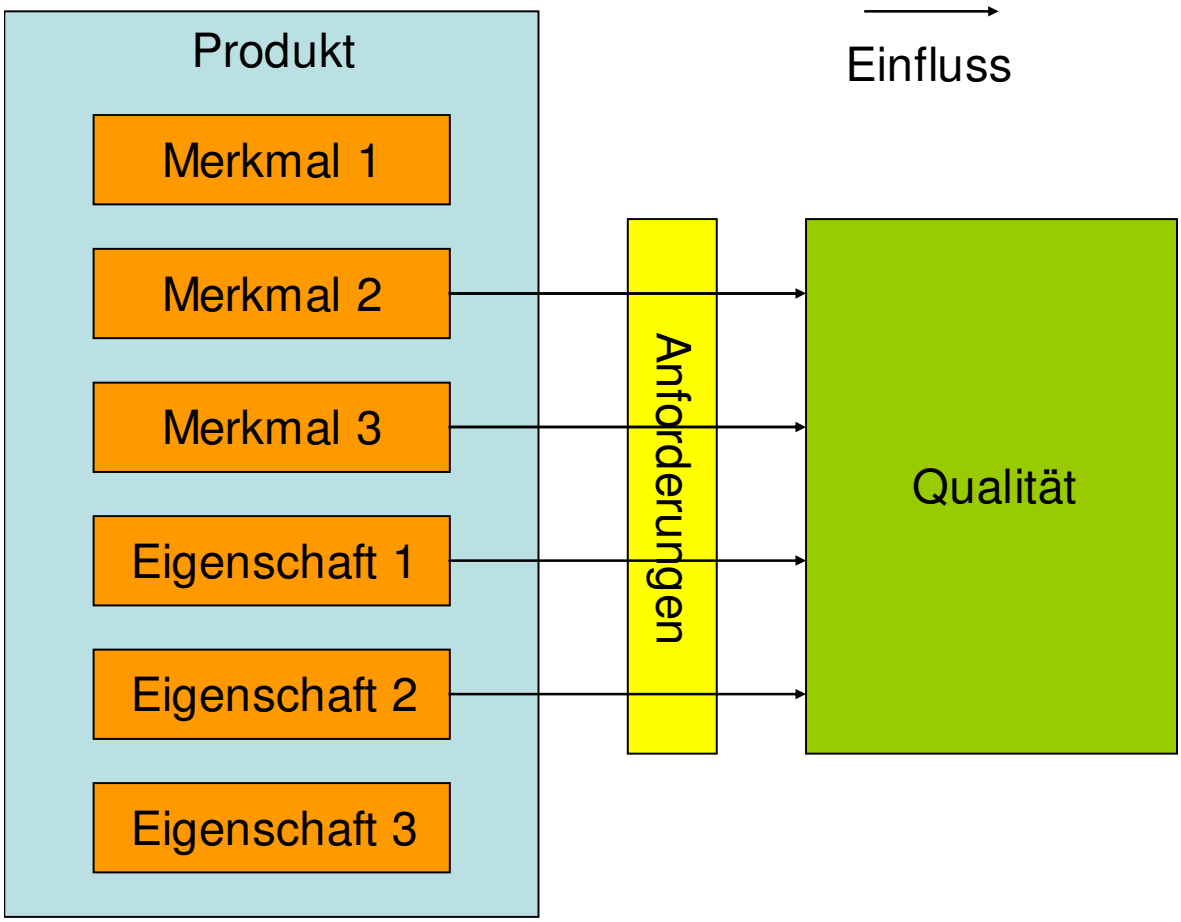


Abbildung 2.11 *Qualität nach ISO 55 350 (eigene Darstellung) (eigene Darstellung)*

Wie in Abbildung 2.11 dargestellt bestimmen also nicht alle Merkmale und Eigenschaften die Qualität (Merkmal 1 und Eigenschaft 3). Erfüllt ein Merkmal oder Eigenschaft die Anforderungen beeinflusst dies die Qualität in positivem Sinne. Wird eine Anforderung durch ein Merkmal nicht erfüllt sink die Qualität.

In der DIN 55 350 wird jede Art von Waren, Rohstoffen, aber auch der Inhalt von Konzepten und Entwürfen als Produkt definiert. Beispiele für Tätigkeiten sind Dienstleistungen, Verfahren und Prozesse (DIN 55 350 Anmerkung 2).

Eine wichtige Aussage enthält die Anmerkung 4 in der DIN 55 350: Die Qualität wird durch die Planungsqualitäten und die Ausführungsqualitäten in allen Phasen des Qualitätskreises (siehe Abbildung 2.12) bestimmt.

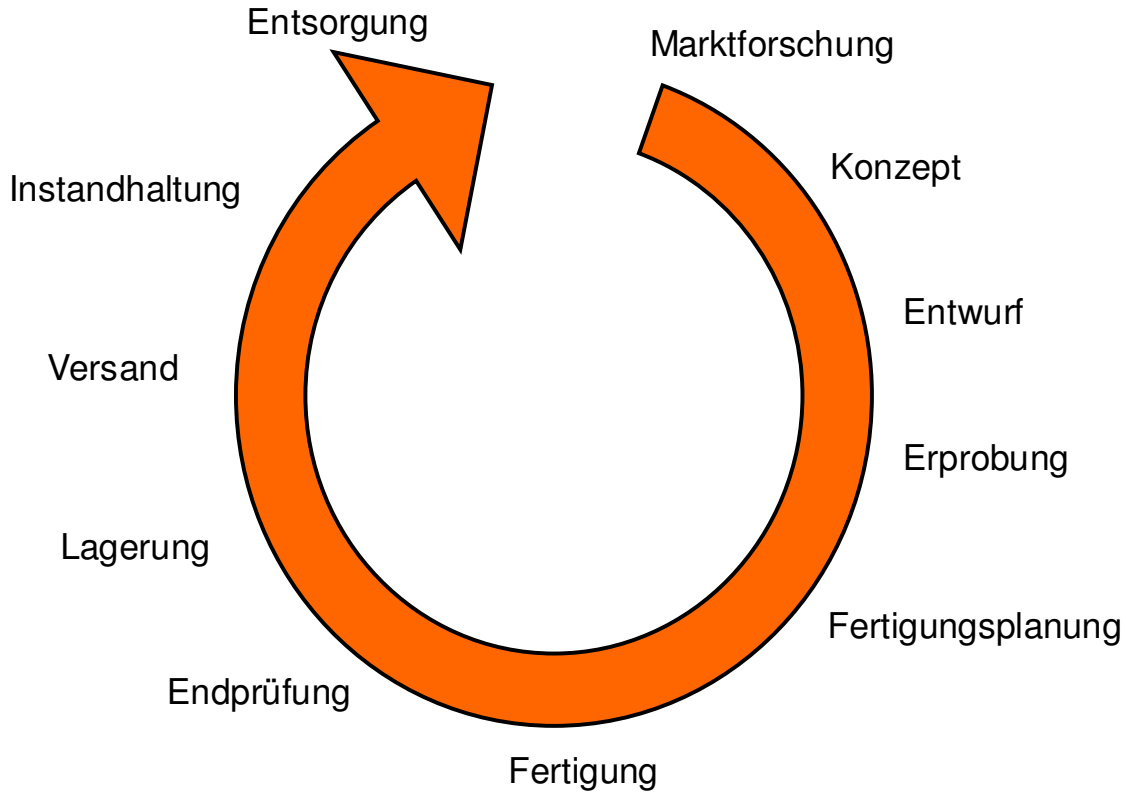


Abbildung 2.12 *Qualitätskreis (nach [ZOL-06, S. 180], eigene Darstellung)*

Die DIN EN ISO 9000:2005 definiert Qualität als den Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt.

2.4. Die Software-Qualität im speziellen

Software-Qualität umfasst mehr, als nur die Erfüllung der Funktionalität. Auch Eigenschaften, wie Wartbarkeit, Effizienz oder Verfügbarkeit zu einem geforderten Termin sind als Teil der Qualität zu sehen (LUD-07 S.63, MOE-07 Teil 1 Seite 37).

Die DIN ISO 9126 definiert Software-Qualität (ähnlich der ISO 55 350) als:

Die Gesamtheit der Merkmale und Merkmalswerte eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen.

Als Qualitätsmerkmale werden in der DIN ISO 9126 sechs Merkmale definiert:

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

In [KAH-98 S.41] werden DIN und ISO[5] Normen genannt, jedoch nicht zur Definition von Software-Qualität herangezogen[6]. Diese wird hier unterteilt in Kriterien aus Benutzer- und Entwicklersicht.

Software-Qualität aus Benutzersicht umfasst demnach:

- Funktionserfüllung
- Effizienz
- Zuverlässigkeit
- Benutzbarkeit
- Sicherheit

Und Software-Qualität aus Entwicklersicht umfasst:

- Erweiterbarkeit
- Wartbarkeit
- Übertragbarkeit
- Wiederverwendbarkeit

Die Benutzersicht wird definiert als die Sicht des Anwenders auf die Software. Für ihn sind alle, mit der Anwendung der Software verbundenen Aspekte von Bedeutung. Als

[5] DIN 40 041 und 55 350, ISO 9126 und 8402

[6] Eine Überschneidung mit den Qualitätsmerkmalen der DIN ISO 9126 ist jedoch offensichtlich

Entwicklersicht wird die Sichtweise betrachtet, die Entwickler auf ein System haben. Dabei spielen die Entwicklung, die Pflege und der Einsatz des Systems eine Rolle.

Des Weiteren wird in KAH-98 darauf hingewiesen, dass die Erstellung von qualitativ hochwertiger Software auch unter wirtschaftlichen Gesichtspunkten zu erfolgen hat.

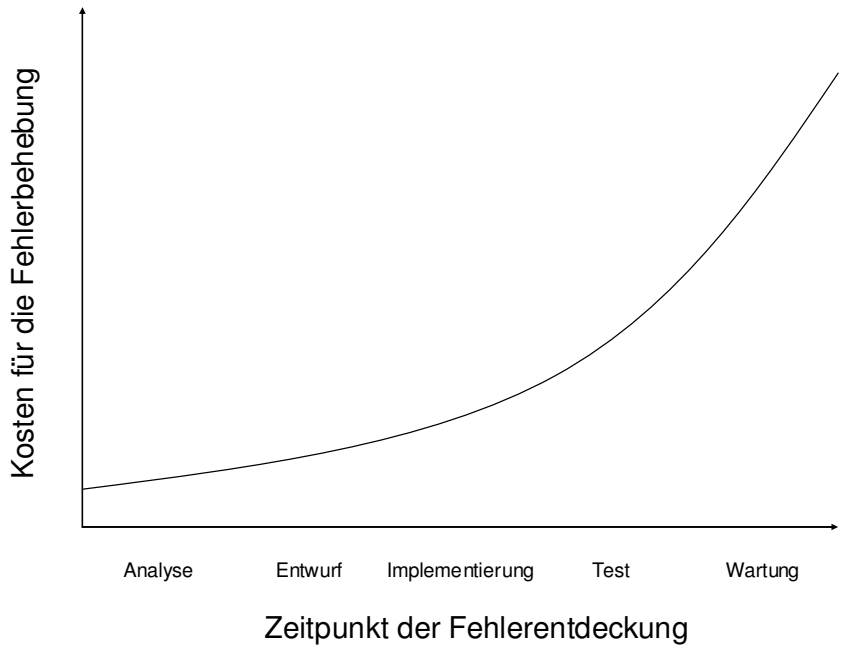


Abbildung 2.13 *Zusammenhang zwischen Behebungskosten und Entdeckungszeitpunkt eines Fehlers ([KAH-98 Seite 43], eigene Darstellung)*

Dies spielt insbesondere bei langfristigen Betrachtungen eine große Rolle. Je später ein Fehler (welcher Art auch immer) in einer Software entdeckt wird, desto teurer ist seine Behebung (Siehe Abbildung 2.13). Werden Fehler in der Analyse oder im Entwurf gemacht und erst in der Wartung entdeck wird die Behebung sehr teuer.

Eine exakte Definition, von dem was Software-Qualität ausmacht kann anscheinend nicht gegeben werden. So schreibt Möller in MOE-07 (Einführung Seite 37), dass Software-Qualität weder exakt gemessen werden kann noch ein exakt definierter Begriff ist. Auch hier wird nochmals auf die Bedeutung der Perspektive hingewiesen, von der das Verständnis des Begriffes Softwarequalität entscheidend abhängt.

Möller definiert folgende Qualitätsmerkmale:

- Korrektheit
- Effizienz
- Robustheit
- Verfügbarkeit
- Zuverlässigkeit
- Benutzerfreundlichkeit
- Datensicherheit und –schutz
- Verständlichkeit und Lesbarkeit
- Änderbarkeit
- Prüf- und Testbarkeit
- Wiederverwendbarkeit und Portabilität

Nur ein Teil dieser Merkmale kann mit einem Werkzeug getestet werden. So wird kein Programm die Wartbarkeit oder die Portabilität testen[7] können.

Ein für diese Arbeit wichtiges Qualitätsmerkmal ist die Prüf und Testbarkeit. Die Testbarkeit ist hoch, wenn der Test reproduzierbar ist. Dazu muss die Ausführung der Prüfung unter definierten Bedingungen stattfinden und eine Auswertung der Ergebnisse vorgenommen werden. [LUD-07 S.63]

Die Qualität von Software ist abhängig von den Faktoren Kosten und Zeit. Je hochwertiger ein System werden soll desto mehr Zeit und Geld wird dafür benötig. Auch die Faktoren Zeit und Kosten sind hängen voneinander ab. Je mehr Geld zur Verfügung steht, desto mehr kann z.B. in gute Entwicklungsumgebungen oder Software von Drittanbieter investiert werden. Dieser Zusammenhang ist selbstverständlich nicht linear und verhält sich in jedem Softwareprojekt anders. Es ist nicht möglich die Entwicklungszeit auf null zu reduzieren, indem man sehr viel Geld in ein Projekt investiert.

[7] Testen im Sinne der in Kapitel 2.6.1 gegebenen Definition

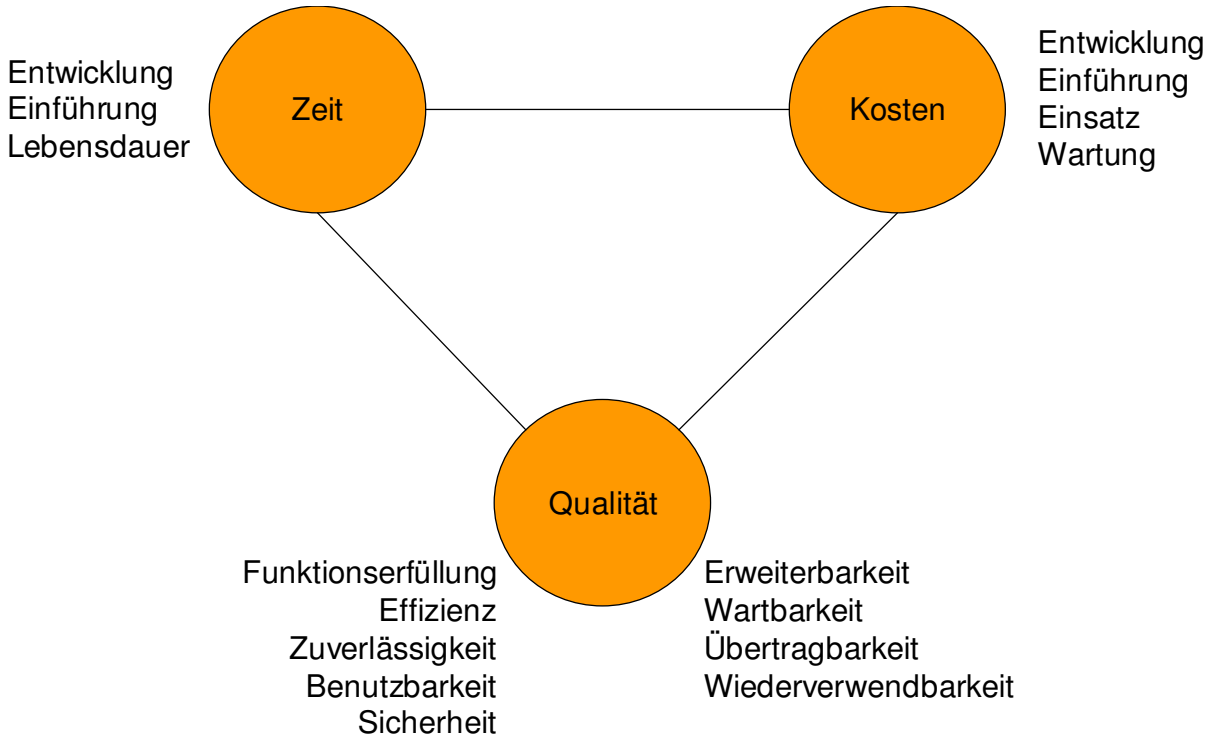


Abbildung 2.14 *Zeit-Kosten-Qualität (nach KAH-98 S. 42)*

Den Zusammenhang zwischen Zeit, Kosten und Qualiät verdeutlicht Abbildung 2.14. Keiner der drei Faktoren kann verändert werden, ohne die anderen zu beeinflussen. Reduziert man die Kosten, sinkt die Qualität und der Zeitbedarf steigt. Will man die Qualität verbessern wird dies nicht ohne zusätzliche Arbeit gehen und damit Zeit und Geld kosten.

2.5. Qualitätsmanagement und Qualitätssicherung

Als Qualitätsmanagement (QM) bezeichnet man nach :

alle Tätigkeiten der Gesamtführungsaufgabe, welche Qualitätspolitik, Ziele und Verantwortung festlegen sowie diese durch Mittel wie Qualitätsplanung, Qualitätslenkung, Qualitätssicherung und Qualitätsverbesserung im Rahmen des Qualitätsmanagementsystems verwirklichen

Alle geplanten und systematischen Tätigkeiten, die innerhalb eines Qualitätsmanagementsystems durchgeführt werden, sind Teil der Qualitätssicherung (QS). Die QS muss dabei so erfolgen, dass Vertrauen in sie geschaffen wird.

Es sei darauf hingewiesen, dass in vielen Normen noch der Begriff Qualitätssicherung statt Qualitätsmanagement verwendet wird [BAL-98 S.278].

Die Software-Prüfung ist der analytische Teil der Qualitätssicherung. Allerdings ist die Prüfung nicht als Hauptbestandteil der Qualitätssicherung zu sehen. Konstruktive und organisatorische Maßnahmen sind wesentlich wichtiger. Sie werden nur durch die Analytischen Maßnahmen ergänzt. [LUD-07 S. 253]

2.6. Der Softwaretest

2.6.1. Definition: Test

Das Wort Test stammt ursprünglich aus dem altfranzösischen und hat seinen Weg über das englische zu uns gefunden [LUD-07 S. 446], [WAH-04]. Es bedeutet soviel wie Erprobung, Versuch oder Prüfung. Es existieren unterschiedliche Definitionen des Begriffes Test. [LUD-07, S. 445] führt zwei Beispiele an:

„Testen ist die Ausführung eines Programms mit dem Ziel, Fehler zu finden“

Und:

„Testen ist die Vorführung eines Programms oder Systems mit dem Ziel zu zeigen, dass es tut, was es tun soll.“

Und Definiert Testen wie folgt:

„Testen ist die – auch mehrfache – Ausführung eines Programms auf einem Rechner, mit dem Ziel, Fehler zu finden.“

Die erste Definition geht auf Myers (1979), die zweite auf Hetzel (1984) zurück. Im Folgenden soll hier Ludewig gefolgt werden. Die Definition von Hetzel zeigt, dass ein Testtool auch zur Präsentation der getesteten Software genutzt werden kann.

2.6.2. Arten von Softwaretests

2.6.2.1. Übersicht

Softwaretests können auf unterschiedliche Arten strukturiert werden. Diese Arbeit folgt im wesentlichen LUD-07 (ab Seite 455).

2.6.2.2. Klassifikation nach Ausgangslage

Man unterscheidet zwischen dem Blackboxtest[8] und dem Whiteboxtest. Führt man einen Blackboxtest durch gibt man vor über den Prüfling nichts zu wissen würde und testet diesen mit unterschiedlichen Eingaben bzw. Testfällen (siehe auch 2.6.3).

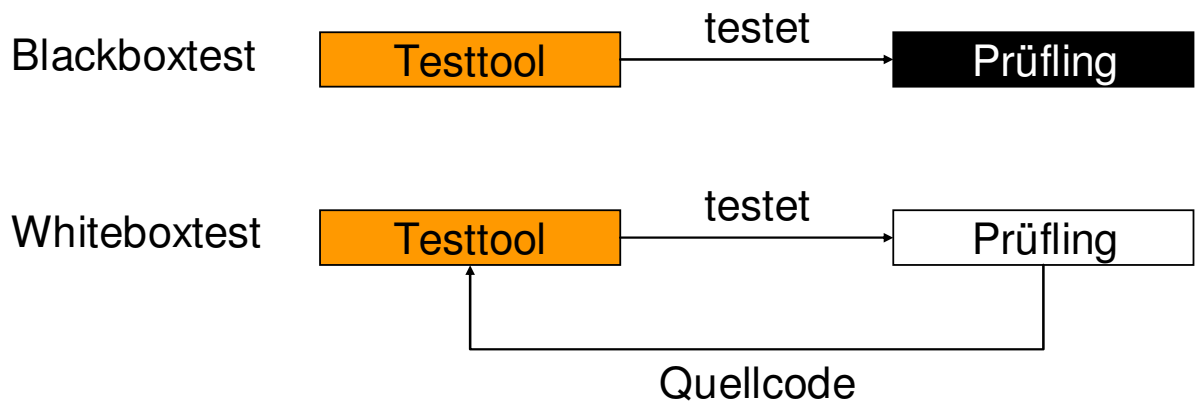


Abbildung 2.15 *Unterschied zwischen Blackbox- und Whiteboxtest (eigene Darstellung)*

Bei einem Whiteboxtest verhält es sich genau umgekehrt: man versucht das Wissen über den Prüfling zu nutzen, um Fehler zu finden [LUD-07 S. 470 – 485], [BAL-98 S. 400 bzw. S. 426]. Whiteboxtests werden oft auch Glassboxtest genannt (siehe z.B. [BAL-98]), sind dies strenggenommen aber nicht. Nach [TES-08] ist eine Glassbox ein System, dessen Eigenschaften zwar offen gelegt sind, dessen Quellcode selber ist jedoch nicht zugänglich ist. Eine Mischung zwischen Whitebox und Blackbox Systemen stellen sogenannte Greybox Systeme dar, deren Eigenschaften nur zum Teil bekannt sind.

[8] Auch Funktionstest genannt

2.6.2.3. Klassifikation nach Komplexität

Eine weitere Art und Weise Softwaretests zu Strukturieren beruht auf der Klassifizierung des Prüflings. Man unterscheidet:

- Einzeltest
- Modultest
- Integrationstest
- Systemtest

Bei einem Einzeltest[9] werden nur kleine Teile eines Programms getestet. Also Beispielsweise Klassen oder Funktionen. Im Modultest[10] wird ein einzelnes Modul bestehend aus mehreren Komponenten (z.B. ein module oder ein package) getestet. Ein Modul muss anschließend in die Software selber integriert werden. Im Integrationstest wird überprüft, ob sich die Klasse auch innerhalb dieser Umgebung erwartungsgemäß verhält und mit ihr korrekt interagiert. Klassischerweise werden hier Fehler in Schnittstellen oder in der Kommunikation aufgedeckt. Im Systemtest wird das Gesamtsystem einer Prüfung unterzogen. Der Systemtest ist damit eigentlich nur ein weiterer Integrationstest, findet jedoch auf einer höheren Ebene statt. Er wird aufgrund des hohen Aufwandes üblicherweise durch ein separates Testteam durchgeführt. [DUS-01 S.413]

Normalerweise wird mit dem Einzeltest begonnen, danach werden Modul-, Integrations- und Systemtest durchgeführt (man geht hier also bottom up vor). [LUD-07 S. 457]

2.6.2.4. Klassifikation nach Aufwand

Klassifiziert man Testverfahren nach dem Aufwand für Vorbereitung und Archivierung, gibt es folgende Testarten:

- Laufversuch
- Wegwerftest
- Systematischer Test

[9] Häufig auch Unit-Test genannt

[10] Auch Kettentest genannt [DUS-01 S.277]

Der Laufversuch wird durch den Programmierer während der Entwicklung durchgeführt. Der Programmierer behebt Probleme und wiederholt den Laufversuch. Bei einem Wegwerftest wird das Programm ohne Vorplanung durch einen Tester (das kann auch der Programmierer sein) gestartet und mit Eingangsdaten gefüttert. Ein Wegwerftest wird nicht archiviert, da seine Qualität aufgrund der zufällig gewählten Testfälle (siehe Kapitel 2.6.3) nicht gesichert ist. Erst der systematische Test wird archiviert. Jemand anderes als der Programmierer leitet aus den Spezifikationen Testfälle ab und testet das Programm mit diesen. Die Ausgaben müssen mit den Spezifikationen verglichen werden. Prüfling, Randbedingungen, Testfälle und Ist-Resultate werden dokumentiert [LUD-07, S. 456]. Der systematische Test ist in der Regel der hochwertigste und zeigt die meisten Fehler auf. Er ist dem Laufversuch oder Wegwerftest fast immer vorzuziehen. Der Laufversuch wird in vielen Fällen trotzdem durch den Programmierer durchgeführt.

2.6.2.5. Klassifikation nach getesteter Eigenschaft

Klassifiziert man Testverfahren nach der Eigenschaft die getestet wird, so erhält man folgende Klassen:

- Funktionstest
- Installationstest
- Wiederinbetriebnahmetest
- Verfügbarkeitstest
- Last- und Stresstest
- Regressionstest

In einem Funktionstest wird getestet, ob ein Prüfling seine Funktion erfüllt. Ob ein Programm installierbar ist und ausschließlich Anforderungen an die Umgebung stellt, die in der Spezifikation vorgegeben sind wird im Installationstest geprüft. In einem Wiederinbetriebnahmetest wird eine Software (z.B. ein LVS) gestoppt und wieder gestartet. Dabei können alte oder inkonsistente Daten (z.B. eine Palette, die ihren Standort verändert hat) zu Problemen führen, mit denen die Software umgehen können muss [ARN-02, S. C4-28]. Im Verfügbarkeitstest wird geprüft, ob eine Software über eine geforderte Zeit hinweg ohne Störungen läuft. Ein Last- oder Stresstest simuliert hohe Belastungen des Systems und testet so, ob die Software auch unter extremen Bedingungen erwartungsgemäß funktioniert

und z.B. Echtzeitanforderungen erfüllt. Je nach Spezifikation wird unter Umständen auch das Verhalten im Überlastbereich getestet. Der Regressionstest testet das Verhalten einer Software nach kleinen Änderungen. Dazu werden die Resultate eines Tests vor der Änderung mit den Resultaten des gleichen Tests nach der Änderung verglichen. Die Ergebnisse sollten, abgesehen von den beabsichtigten Änderungen, identisch sein [LUD-07, S.457].

2.6.2.6. Klassifikation nach beteiligten Rollen

Im laufe der Entwicklung testen unterschiedliche Gruppen die Software. Man spricht von

- Alphatests
- Betatests
- Abnahmetests

Alphatests bezeichnen alle Tests, die bei einem Hersteller durchgeführt werden. In einer zweiten Phase, dem Betatest, wird die Software zukünftigen Nutzern zur Verfügung gestellt. Im Abnahmetest[11] wird dem Kunden die Software vorgeführt. Dies ist also kein Test im klassischen Sinne, sondern eher eine Demonstration [LUD-07, S.458].

2.6.2.7. Klassifikation nach Automatisierungsgrad

Ein Test kann sowohl manuell, als auch automatisch durchgeführt werden. Allerdings sind nicht alle Tests automatisierbar (z.B. die Prüfung eines ausgedruckten Papiers). Manuelle Tests sind fehleranfälliger, automatisierte Tests dagegen erfordern leistungsfähiges Personal, das organisiert und strukturiert vorgeht [DUS-01 S.29]. Um das automatisierte Testen zu unterstützen existieren diverse Tools (siehe Kapitel 2.6.6).

Um die Vorteile von automatisierten Tests nutzen zu können, müssen diese korrekt implementiert werden. Es muss also ein gewisser Aufwand betrieben werden, bevor die Vorteile des automatisierten Testens genutzt werden können.

[11] Auch Akzeptanztest genannt

Diese sind:

- Erstellung eines zuverlässigen Systems
- Verbesserung der Testqualität
- Verringerung des Testaufwandes und Reduzierung des Zeitplans

[DUS-01 S.44/45].

2.6.2.8. Klassifikation nach Abdeckung

Man kann Testverfahren klassifizieren, die bestimmte Bedingungen erfüllen:

- Anweisungsabdeckung
- Entscheidungsabdeckung
- Bedingungsabdeckung
- Pfadabdeckung
- Datenflussabdeckung
- Verzweigungsabdeckung

Wird durch einen Test jede Anweisung im Prüfling mindestens einmal ausgeführt, spricht man von Anweisungsabdeckung. Der Test auf Anweisungsabdeckung wird auch C1 genannt. Von Entscheidungsabdeckung spricht man, wenn alle theoretisch denkbaren Entscheidungsergebnisse ausgeführt wurden. Man nennt solche Tests auch Verzweigungsabdeckungstests, sie werden als C2 bezeichnet.

Bedingungsabdeckung[12] unterscheidet sich nur darin von der Entscheidungsabdeckung, dass der Test die Korrektheit der wahren oder falschen Ergebnisses jedes booleschen Ausdrucks nachweist.

Werden im Laufe eines Tests alle im Programm möglichen Pfade mindestens einmal durchlaufen spricht man von Pfadabdeckung. Diese kann bei Schleifen (die theoretisch unendlich oft durchlaufen werden können) nicht sichergestellt werden. Die Zahl der möglichen Pfade kann sehr hoch sein, da sie sich exponentiell zur Anzahl der

[12] Auch Termüberdeckung [LUD-07 S.481]

Verzweigungen verhält. So ist es sehr zeitintensiv Pfadabdeckung zu erreichen. Daher beschränkt man sich auf eine begrenzte Anzahl von Durchläufen. Darüber hinaus sind pfadabdeckende Tests sehr gründlich, daher empfiehlt sich der Einsatz dieser Testart für erfolgskritische Funktionen. Laut Ludewig [LUD-07 S.481] spielt Pfadabdeckung in der Praxis keine Rolle.

Von Datenflussabdeckung wird gesprochen, wenn der Datenfluss in das Testverfahren einbezogen wird und Pfadabschnitte ausgewählt werden, die Merkmale des Datenflusses für alle möglichen Typen von Datenobjekten erfüllen.

Verzweigungsabdeckung[13] liegt vor, wenn alle logischen Verzweigungen mit wahren und falschen Bedingungen ausgeführt wurden.

Diese Testverfahren können nur in einem Whitebox Test angewandt werden. [DUS-01 S.275-280] und [LUD-07 S.481]

2.6.3. Testfälle

Als Testfall bezeichnet man eine Situation in der eine Software getestet werden soll. Dazu gehören:

- Anfangszustand von Umgebung und Software
- die Werte der Eingabedaten
- alle notwendigen Bedingungen
- die erwartete Ausgabe

Testfälle sollten so gewählt werden, dass die Wahrscheinlich einen Fehler zu entdecken möglichst hoch ist [LUD-07 S. 464]. Ein idealer Testfall ist demnach:

- repräsentativ (er deckt möglichst viele andere Szenarien ab)
- fehlerintensiv (hat eine hohe Wahrscheinlichkeit einen Fehler aufzudecken)
- redundanzarm (er prüft nicht, was andere Testfälle bereits prüfen)

[13] Auch Zweigüberdeckung genannt [LUD-07 S.481]

Um zu verhindern, dass viele Testfälle redundant sind, kann man mögliche Testfälle in Äquivalenzklassen einteilen und dann aus jeder Äquivalenzklasse einen Fall auswählen.

Beispiel

Eine Eingabevariable darf als Werte ganze Zahlen von 0 bis 100 annehmen. Es ergeben sich folgende Äquivalenzklassen:

Äquivalenzklasse	Beispielhafte Werte	Grenzfälle	Repräsentanten
Gültige Zahlen	0,1,…, 99, 100	-1,0,100,101	0,52,100
Ungültige Ganze Zahlen	-100,-42,142		-100, 200, -1, 101
Ungültige reale Zahlen	-5.12, 2.56, 1.024, 100.2		-5.12, 2.56, 100.2

Tabelle 2.6 *Beispiel für Äquivalenzklassen von Testfällen*

In diesem Beispiel wurde die Annahme getroffen, dass der Benutzer ungültige Werte selber eingibt. Reale Zahlen müssen nicht getestet werden, wenn nur ein Schieber zur Verfügung steht, dessen Ausgabe ohnehin nur ganze Zahlen sind. Auch wurde angenommen, dass nur diese eine Variable zur Verfügung steht. Eine besondere Bedeutung kommt den Grenzen zwischen gültigen und ungültigen Werten zu.

2.6.4. Softwaretest und –prüfung

2.6.4.1. Grundsätzlicher Testablauf

Ein systematischer Test ist (wie in Kapitel 2.6.2.4 dargestellt) Laufversuchen oder Wegwerftests vorzuziehen. Im Kern besteht der systematische Test aus Vorbereitung, Ausführung und Auswertung. Planung und Analyse erfolgen vor bzw. nach dem eigentlichen Test und sind ebenso wichtig. Der Grundsätzliche Ablauf und die zu erstellenden Dokumente sind in Abbildung 2.16 dargestellt. Die Prüfung eines komplexen Systems besteht im Allgemeinen aus vielen einzelnen Test und beansprucht erhebliche Resourcen.

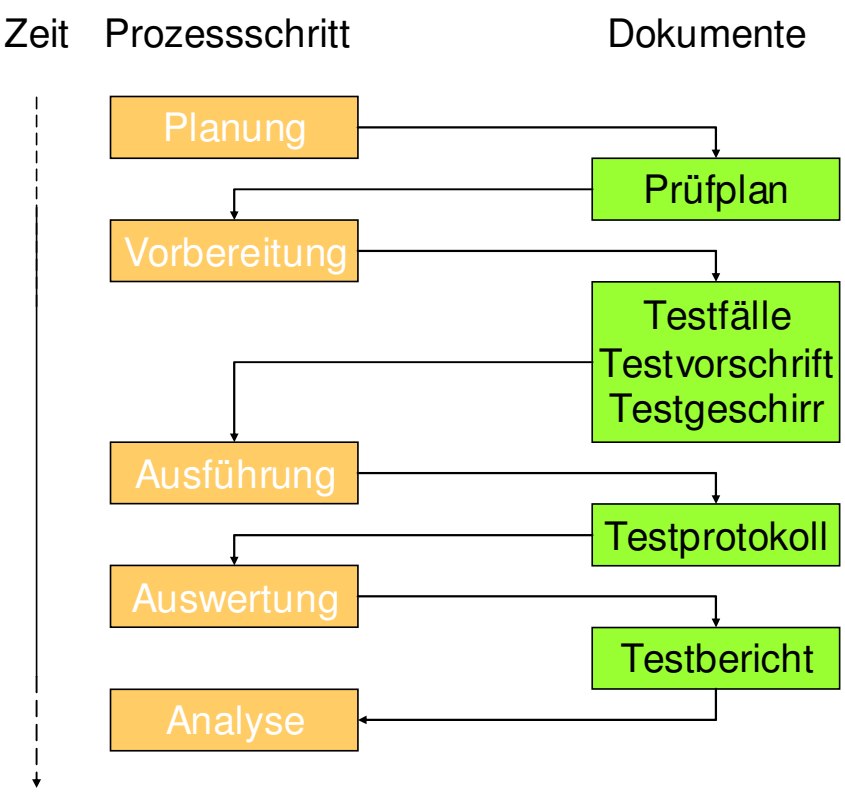


Abbildung 2.16 *Grundsätzlicher Testablauf (nach [LUD-07] eigene Darstellung)*

Daher ist eine sorgfältige Planung unumgänglich. In dieser wird festgelegt, welche Eigenschaften in welchem Test geprüft werden sollen. Zusätzlich werden Testziele und –Strategien bestimmt. Ausgehend von diesen wird jeder einzelne Test geplant. Dabei müssen benötigte Resourcen und die Dauer für die man die Resourcen benötigt ermittelt werden. Nach Abschluss des Projektes werden die Testberichte analysiert. Dabei wird überprüft, ob die gewählten Strategien und Aufwände sinnvoll auf die Aufgabe abgestimmt waren. Die Ergebnisse der Analyse können in Folgeprojekten Verwendung finden. [LUD-07 S.459]

2.6.4.2. Vorbereitung

Den Test vorzubereiten ist der aufwändigste Teil des gesamten Tests. Wird hier sehr viel Aufwand getrieben kann der Test selber sogar automatisiert werden (siehe auch Kapitel 2.6.5). In der Vorbereitung findet auch die Spezifikation der Testfälle statt. Des Weiteren benötigt man ein Testgeschirr, bestehend aus Testtreiber[14] (sorgt für Eingaben), Platzhalter[15] (steht für Komponenten, die der Prüfling nutzt, die aber noch nicht implementiert sind) und zusätzlichen Werkzeugen wie z.B. Monitoren, die Schnittstellen beobachten (siehe Abbildung 2.17). Das Testgeschirr muss in der Vorbereitungsphase bereit gestellt werden. Insbesondere bei sicherheitskritischen Anwendungen ist die Bedeutung des Geschirrs enorm

[14] Engl. Testdriver

[15] Engl. stub

wichtig und sollte daher genauso sorgfältig erstellt und gewartet werden, wie der Prüfling [LUD-07, S. 460].

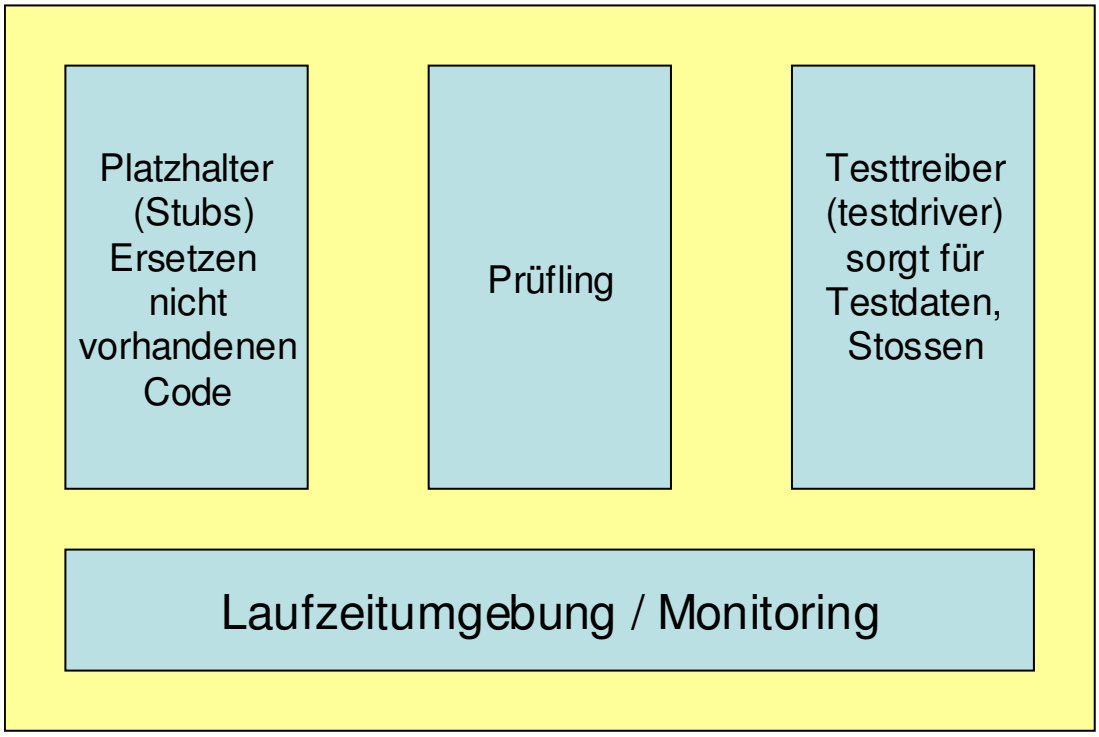


Abbildung 2.17 *Testgeschirr (nach [PAE-04] S.5, eigene Darstellung)*

2.6.4.3. Ausführung

Während der Ausführung des Tests wird protokolliert, welches Verhalten der Prüfling in den einzelnen Tests gezeigt hat. Wichtig ist, dass der Prüfling nicht während des Tests verändert wird, damit genau eine Version des Prüflings getestet wurde und der Programmierer hinterher die Fehler beheben kann. Solche Änderungen können den Fehler verdecken anstatt ihn zu beheben oder sogar neue Fehler an anderer Stelle hervorrufen. Im schlimmsten Fall ist der Test, der einen solchen Fehler entdeckt hätte, bereits durchgeführt worden. Der Fehler bleibt dann unentdeckt.

Für die Testausführung werden folgende Anforderungen an den Prüfling, den Test und den Tester gestellt:

- Keine speziellen Versionen des Prüflings für den Test
- Kein Abbruch des Tests, wenn Fehler erkannt wurden

- Keine Modifikation während des Tests
- Kein Wechsel zwischen Test und Debugging

Es ergeben sich daraus die folgenden Vorteile:

- Der Testaufwand lässt sich relativ gut abschätzen
- Der Test deckt die grundsätzlichen Mängel auf
- Es werden keine kumulativen Korrekturen durchgeführt
- Das Testgeschirr wird effizient genutzt
- Der Prüfling wird nicht während des Tests beschädigt (z.B. durch Zusätze, die zur Fehlersuche eingefügt wurden)
- Die Ergebnisse beziehen sich auf eine bestimme Version eines bestimmten Programms und sind damit nachvollziehbar

Das Protokoll sollte der Konfigurationsverwaltung unterstellt sein, da es den Test für alle Beteiligten nachvollziehbar macht [LUD-07 S.461].

2.6.4.4. Auswertung

In der Auswertung werden die Resultate des Tests mit den Soll-Vorgaben verglichen. In automatisierten Tests wird dieser Vorgang durch ein Werkzeug unterstützt oder übernommen. Das Ergebnis ist der Testbericht, alle den Test betreffenden Dokumente werden in ihm aufgeführt. Außerdem sind alle nötigen administrativen Angaben (z.B. beteiligte Personen, gefundene Fehler, usw.) aufzuführen. Am Ende des Testberichts wird festgehalten, ob das Testendekriterium erreicht wurde [LUD-07, S.462].

2.6.5. Der automatische Softwaretest

In den Anfängen der Softwareentwicklung wurden Testverfahren auf Papier festgehalten. Die Tests zielten darauf ab Fehler in drei unterschiedlichen Klassen aufzudecken:

- Fehler in Steuerflüssen
- Fehler in Algorithmen
- Fehler in der Datenbearbeitung

Tests wurden ausschließlich am Ende von Projekten durchgeführt. Dies hatte seine Ursachen darin, dass der größte Teil der damals entwickelten Software umfangreiche Programme für Forschung und Verteidigung war, die auf Großrechnern oder Minicomputerplattformen lief.

Mit dem Aufkommen des PCs wurden andere, kommerzielle Softwareprojekte entwickelt. Aufgrund der Kommerzialisierung wuchs der Druck auf die Entwickler, schneller und effizienter zu werden. Zusätzlich hatten sich die Programme verändert: Während zu Anfang die Stapelverarbeitung vorherrschend war, nutzten die Anwender nun Onlinesysteme. In diesen können Benutzer Jobs in einer beliebigen Reihenfolge aufrufen. Die Rahmenbedingungen hatten sich also in zweierlei Hinsicht geändert:

- zum einen musste der Test effizienter werden
- zum anderen resultierte aus den Programmen mit graphischer Benutzerschnittstelle eine höhere Komplexität des Prüflings. Also mussten auch die Tests angepasst werden.

Das automatische Testen kann dabei helfen effizient und gründlich zu prüfen. [DUS-01 S. 4] definiert den automatischen Test als:

Verwaltung und Durchführung von Testaktivitäten einschließlich der Entwicklung und Ausführung von Testskripts zur Überprüfung der Testanforderungen mit Hilfe eines automatischen Testwerkzeugs“

2.6.6. Unterstützung des Testens durch Tools

Es existieren unterschiedliche Tools, um verschiedene Tests durchzuführen und zu unterstützen. Zunächst unterscheidet man auch hier wieder zwischen Black- und Whitebox Testtools (siehe Kapitel 2.6.2.2). Des Weiteren existieren verschiedenartige Werkzeuge (Auswahl):

- Generatoren für Testverfahren
- Werkzeuge für das Testmanagement
- Werkzeuge zum Testen/Bedienen von graphischen Oberflächen

- Werkzeuge zum Testen von Last, Leistung und Belastung
- Werkzeuge zum Erzeugen produktbasierter Testverfahren
- Testdatengeneratoren
- Hilfsprogramme zum Vergleichen von Dateien
- Simulationswerkzeuge

Generatoren für Testverfahren erzeugen zufällige Eingabeszenarien. Diese werden benutzt, um Tests mit willkürlichen Eingaben durchzuführen. Zusätzlich gibt es Generatoren, die in der Lage sind auf alte Daten zurückzugreifen, um typische Fehler aufspüren zu können. Solche Programme sind also in der Lage Wissen aufzubauen.

Tools, die das Testmanagement unterstützen bieten z.B. Funktionalitäten, um den Testlebenszyklus zu planen, zu verwalten oder zu analysieren. Als Beispiel sei hier Zephyr angeführt (Abbildung 2.18).

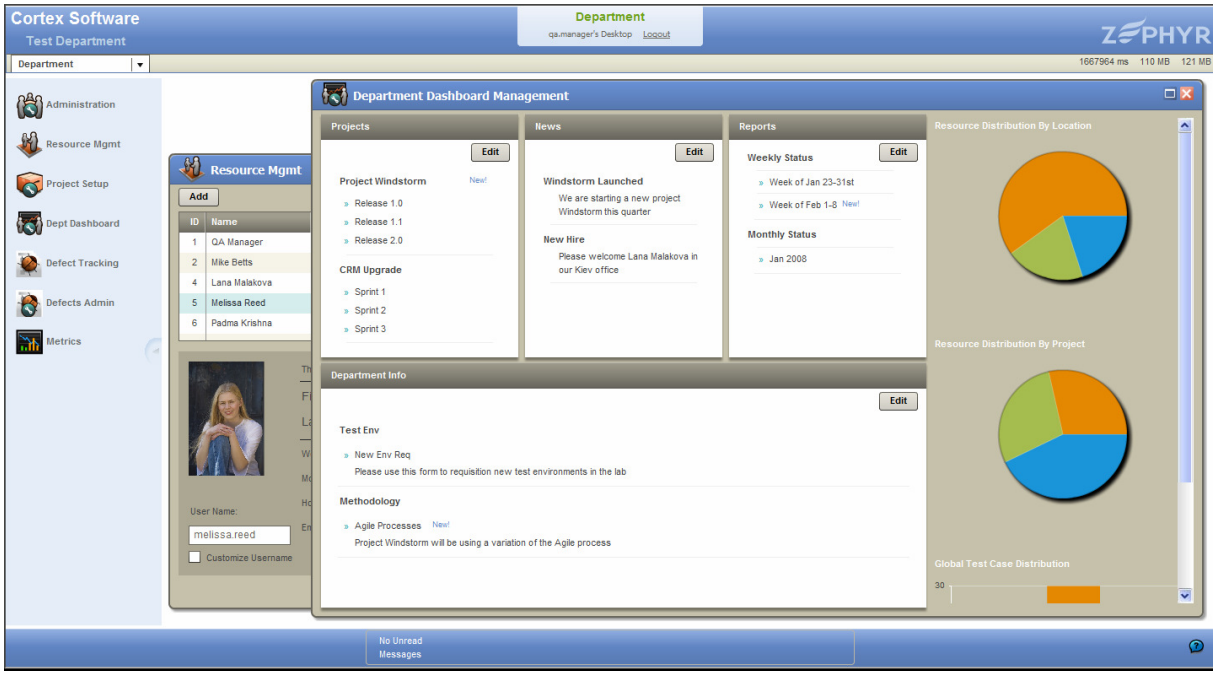


Abbildung 2.18 *Zephyr, ein Test Management Tool [ZEP-08]*

Werkzeuge, die graphische Oberflächen bedienen können nennt man auch Record/Replay- oder Capture/Replay-Werkzeuge. Mit ihnen ist es möglich die Bedienung eines GUIs (graphical user interface) aufzuzeichnen und wiederzugeben. Die Aufzeichnung liegt als Quellcode vor und kann benutzt werden, um ein Testprogramm zu entwickeln.

Leistungs- Belastungs- und Lasttests können durch Tools unterstützt werden, die z.B. in Client/Server Systemen viele Anfragen an den Server erzeugen. Diese Tools sind in der Lage umfangreiche Statistiken zu erzeugen. SpaceNet_Lasttest ist ein solches Tool (siehe Abbildung 2.19).

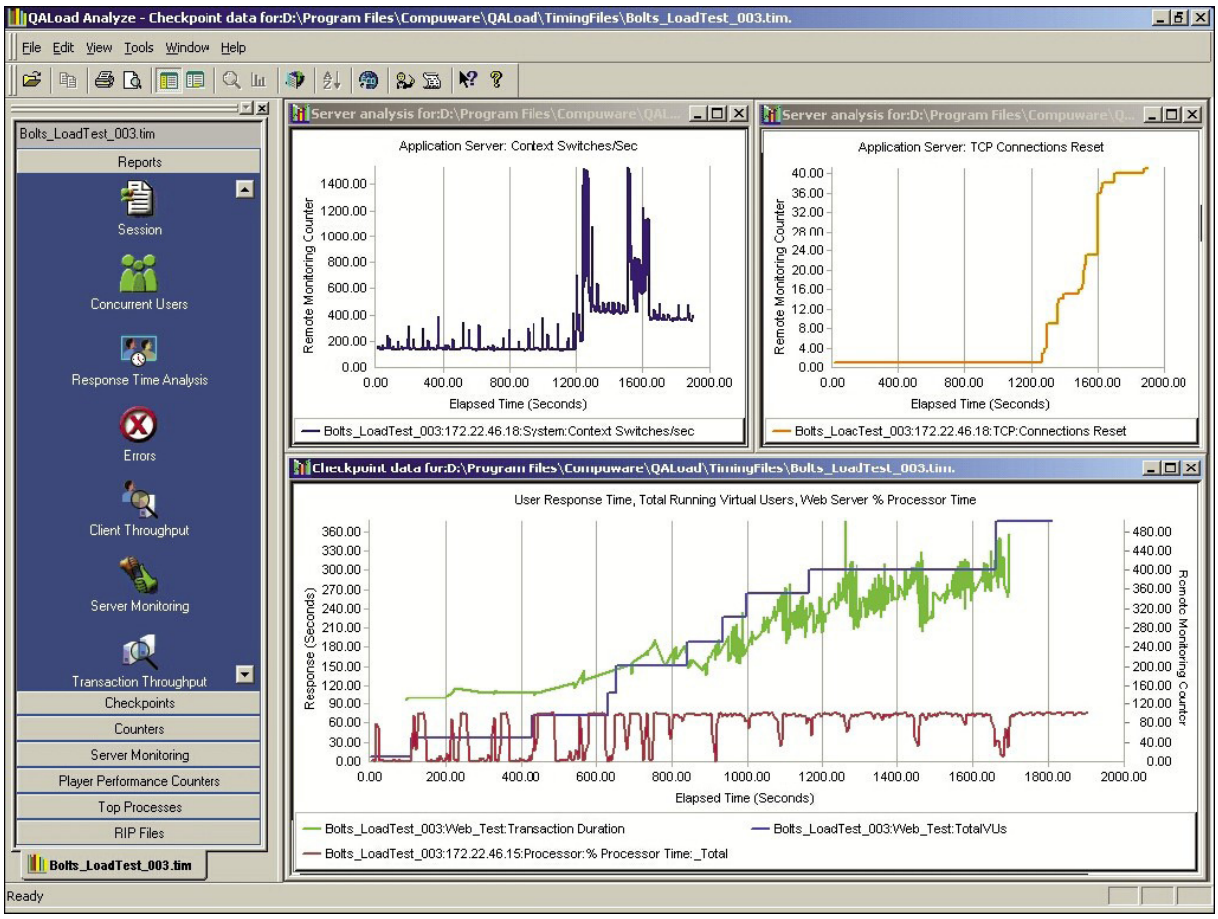


Abbildung 2.19 *SpaceNet_Lasttest Lasttestool [SPA-08]*

Testverfahren lassen sich durch Programme aus dem Prüfling ableiten. Dabei wird versucht Pfadüberdeckung bzw. Anweisungsüberdeckung zu erreichen, damit große Teile des Programms ausgeführt und damit getestet werden.

Testdatengeneratoren dienen dazu große Mengen sinnvoller Daten zu produzieren, mit denen z.B. Datenbanken gefüllt werden können. Manche Testdatengeneratoren verfügen darüber hinaus über Fähigkeiten

- csv oder Excel Dateien zu erzeugen
- die Tester beim Informationsaustausch zu unterstützen
- Schlüssel zu vereinheitlichen

Simulationswerkzeuge können das Testen in vielerlei Hinsicht unterstützen. Zum Beispiel, indem man mit ihnen andere Anwendungen oder reale Objekte mit denen kommuniziert oder interagiert werden soll modelliert. So besteht die Möglichkeit Modelle von Dingen zu erstellen, die noch gar nicht existieren.

Mit einem Simulationstool ist es möglich unterschiedliche Varianten zu testen („Was wäre wenn …“) oder Leistungsdaten zu messen. Mit einer Simulation ist es möglich das Verhalten und die Logik des modellierten Systems nachzubilden. Die Simulation kann so ein sinnvolles Verhalten gegenüber dem Prüfling an den Tag legen. [DUS-01 S.98ff]

2.6.7. Grenzen von Softwaretests

Abgesehen davon, dass ein Test nicht alle Qualitätskriterien (siehe auch Kapitel 2.4) prüfen kann, ist es nicht möglich umfangreichere Software vollständig zu testen. Ein kleines Beispiel macht dies eindruckvoll klar:

Ein Programm mit drei Eingabeparametern mit je 32 bit hat 2^{96} oder etwa $8*10^{28}$ mögliche Startzustände und zur vollständigen Prüfung damit genauso viele Testfälle. Ist man in der Lage pro Sekunde eine Milliarde (10^9) Testfälle zu testen, so braucht man etwa 2.5 Billionen Jahre oder etwa das 180-fache Alter des Universums um alle Tests durchzuführen:

- Alter des Universums: ~14 Millarden Jahre
- $8*10^{28}$ Testfälle / (10^9 Testfälle/s) = $8*10^{19}$ Sekunden
- ein Jahr hat 60 * 60 * 24 * 365 ≈ $3.15 * 10^7$ Sekunden
- $8*10^{19}$ s / $3.15*10^7$ s/a ≈ $2.5 * 10^{12}$a

Daraus folgt: $2.5*10^{12}$a / $14*10^9$a ≈ 180. (Idee nach [LUD-07 S. 453], leicht verändert). Ein ähnliches Beispiel findet sich in [DUS-01 S.42/43].

Dijkstra hat daher schon 1970 folgenden berühmten Satz formuliert:

Program testing can be used to show the presence of bugs,
but never to show their absence![16]

Das Testen von Software kann also in keinem Fall nachweisen, dass ein Programm fehlerfrei ist. Die Qualität von Software kann aber durch geschicktes Testen durchaus verbessert werden.

3. Konzept zum Testen von Lagerverwaltungssystemen

3.1. Überblick

Lagerverwaltungssysteme interagieren mit unterschiedlichen Partnern (siehe Kapitel 2.1.2.4 und
Abbildung 2.6). Um Fehler in einem LVS zu finden kann man es also von „mehreren Seiten“ aus testen. Zum einen kann man die graphischen Schnittstellen, die die Interaktion mit dem Lagerleiter und seinen Mitarbeitern ermöglichen, testen. Zum anderen muss auch die Kommunikation mit dem Lager getestet werden. Ein Testen der Interaktion des Lagerverwaltungssystems mit anderer Software (z.B. mit einem ERP System wie SAP) ist zusätzlich notwendig.

Diese Arbeit beschränkt sich weitgehend darauf, das Testen des LVS durch eine Simulation zu untersuchen, geht aber in Kapitel 3.3 auch kurz auf andere Testmöglichkeiten ein.

Der Test eines LVS verläuft zunächst ähnlich jedem anderen. Es müssen also Tests geplant, vorbereitet, durchgeführt, ausgewertet und die Berichte analysiert werden. Betrachtet man den Einzel-, Modul- oder Integrationstest, so können diese Tests genau so wie in anderen Softwareprojekten ablaufen, eine Unterstützung durch eine Simulation ist hier nicht unbedingt erforderlich. Will man jedoch das Verhalten des LVS im Systemtest prüfen, steht man vor dem Problem, dass man das Verhalten eines Lagers bzw. MFRs abbilden muss um dem LVS sinnvolle Antworttelegramme senden zu können. Ohne diese ist ein Test nicht durchführbar. Es ergeben sich mehrere denkbare Varianten des Systemtests:

[16] Ein Programmtest kann genutzt werden, um die Anwesenheit von Fehlern, nicht um ihre Abwesenheit zu zeigen

- Mithilfe von Telnet (nur bei TCP/IP basierter Kommunikation, aufwändig, wenig Sinnvoll)
- Indem man die Kommunikation zwischen einer anderen Installation des LVS und einem bereits bestehenden Lager „mitschneidet“ und die Antworten des Lagers nutzt, um einen Regressionstest durchzuführen (bedingt geeignet, da keine zwei Läger identisch sind und die Anpassungen so eventuell nicht getestet werden können)
- Mit einem Tool:
    - Simulation: Abbildung des Lagers in einem Simulationstool
    - TCP Test Tools (Tools, die TCP Pakete versenden und empfangen)
    - Andere Tools (z.B. Lasttesttools, um die Belastbarkeit der Datenbank zu testen)
- Indem man das LVS mit dem realen Lager testet.

Der manuelle Test eines LVS durch die Eingabe von Telegrammen per Hand ist langsam und aufwändig. Das Mitschneiden der Kommunikation mit einem bereits existierenden Lager wird in den meisten Fällen ausscheiden, da man dazu zwei nahezu identische Läger benötigen würde. Ein Regressionstest baut auf ein Programm auf, an dem nur kleine Änderungen vorgenommen wurden. Unterscheiden sich jedoch die Läger .stark voneinander, wird auch der Datenverkehr zwischen Lager und LVS anders sein.

Der Test am „lebenden“ Lager wird kurz vor Inbetriebnahme durchgeführt und ist extrem kostenintensiv (siehe Kapitel 1.2). Um diese Kosten zu senken bietet es sich an Tools (wie z.B. eine Simulation) zu nutzen. Ein Fehler der in einem Test gefunden und erfolgreich behoben wurde wird im Betrieb nicht mehr auftreten.

3.2. Test eines LVS mit einer Simulation

3.2.1. Vorgehensweise

Das prinzipielle Vorgehen zum Testen eines LVS wird sich nicht maßgeblich von herkömmlichen Softwaretests unterscheiden. Die grundsätzlichen Schritte werden also auch hier Planung, Vorbereitung, Durchführung, Auswertung und Analyse sein. Wobei der Test mit einer Simulation nur als ein Teil des gesamten Testprogramms zu betrachten ist. Planung und Analyse finden also in einer übergeordneten Ebene statt.

Um die Tests zu automatisieren, kann es notwendig sein, ein Tool zu benutzen, das graphische Schnittstellen bedient (z.B. um das LVS in einen definierten Zustand zu versetzen).

Testet man ein LVS mit einer Simulation, so muss diese Simulation neben dem Lager auch die Schnittstellen des Lagers (z.B. den Materialflussrechner oder das Staplerleitsystem) abbilden. Es müssen also Module vorhanden sein, die in der Lage sind, über das Netzwerk mit dem LVS zu kommunizieren. Wird beispielsweise der Safeline Standard eingesetzt, (siehe Kapitel 2.2.2) muss die Simulation mit dem LVS über zwei TCP Verbindungen kommunizieren.

Damit der Test reproduzierbar ist müssen alle Eingangsdaten, sowie der Zustand von Simulation und LVS zu beginn des Tests erfasst werden.

3.2.2. Testarten

Der Test des LVS durch eine Simulation kann als Black- oder Glassboxtest durchgeführt werden. Ein Glassboxtest ist allerdings nur möglich, wenn der Quellcode des Systems vorliegt. Will der Abnehmer (Lagerbetreiber) das LVS mit einer Simulation im Abnahmetest prüfen, kommt diese Art des Testens also nicht in Frage.

Laufversuche werden sicher durch den Programmierer durchgeführt, können aber nicht als ausreichend angesehen werden. Aufgrund der Bedeutung des Tests und der Komplexität eines LVS ist davon abzusehen Wegwerftests durchzuführen. Es bleibt also der systematisch Test, der in Kapitel 2.6.2.4 vorgestellt und als hochwertigster eingestuft wurde.

Ein LVS muss nicht nur auf Funktionalität und Installierbarkeit, sondern auch auf Belastbarkeit und Wiederinbetriebnahme getestet werden. Insbesondere der Test auf Wiederinbetriebnahme ist heikel, da sich der Zustand des Lagers während der Offline Phase ändern kann.

Beispiel

Nach einem unvorhergesehenen Ereignis (z.B. Hardwaredefekt) muss das LVS eines Ersatzteillagers seinen Betrieb einstellen. Ein wichtiger Kunde benötigt aber dringend einige

Ersatzteile. Daher werden diese (ohne sie im Lagerverwltungssystem zu verbuchen) in aller Eile zusammen- und dem Kunden zugestellt. Des Weiteren hat sich der Standort von anderen Teilen z.B. auf einer Fördertechnik verändert. Nachdem die Sendung das Lager verlassen hat ist das Lagerverwaltungssystem wieder verfügbar. Der Zustand des LVS stimmt nun nicht mehr mit dem der Realität überein. Es ist notwendig die eilig zusammengestellte und abgeschickte Sendung des wichtigen Kunden nachträglich zu verbuchen und die geänderten Standorte der Teile auf der Fördertechnik zu erfassen.

Ein Test auf Verfügbarkeit, der prüft, ob das LVS die geforderte Zeit ohne Störungen läuft, ist nur eingeschränkt möglich, da diese Dauer im Allgemeinen mehrere Jahre beträgt.

Last- und Stresstests sind angebracht, da ein Lager erweitert werden kann und somit die Last des LVS steigt. Hierfür kann eine Simulation sinnvoll eingesetzt werden. Man lässt die Simulation schneller laufen[17] und erzeugt so erheblich mehr Datenverkehr. Dieser muss durch das LVS bearbeitet werden und stellt damit eine erhöhte Last dar.

Sollten Änderungen am LVS vorgenommen werden kann mit einer Simulation ein Regressionstest (siehe Kapitel 2.6.2.5) durchgeführt werden. Wird das LVS durch ein Konkurrenzprodukt ersetzt, kann auch ein Regressionstest durchgeführt werden.

Will man Einheiten- Modul- oder Integrationstests mit einer Simulation durchführen muss entsprechendes Testgeschirr zur Verfügung gestellt werden, das nicht vorhandene oder noch nicht angepasste Teile des LVS ersetzt.

3.2.3. Grenzen

Mit einem Simulationstool lässt sich nur die Funktionalität des LVS gegenüber dem Lager testen. Die Benutzung der graphischen Schnittstellen und die Kommunikation mit weiterer Software muss an anderer Stelle geprüft werden. Alle Funktionen, die die Beteiligung von Anwendern erfordern, können nur ausreichend getestet werden, indem die entsprechenden Schnittstellen bedient werden. Dies kann und soll nicht durch die Simulation, sondern durch dafür vorgesehene Werkzeuge (siehe Kapitel 2.6.6) geschehen. Trotzdem kann es nötig sein, dass die Simulation Teile der Clients (z.B. am Kommissionierplatz) mit simuliert. Dies

[17] Simulationszeit/realer Zeit > 1

ist nötig, damit das LVS nicht auf Eingaben wartet und das Material (z.B. am Kommissionierplatz) weitergeleitet wird.

Betreibt man ausreichend Aufwand, kann man ein Lager recht wirklichkeitsnah mit einem Simulationstool nachbilden. Grenzen sind dem Testen mit Simulationstools also vor allem durch die Auswahl und Anzahl der Testfälle gesetzt.

Alle Funktionen, die in irgendeiner Weise Daten ausdrucken, können nicht automatisch getestet werden, da der Ausdruck nicht mit vertretbarem Aufwand automatisch zu erfassen und zu verarbeiten ist. Alle Ausdrucke müssen also manuell auf ihre Richtigkeit geprüft werden. Wie bereits in Kapitel 2.6.7 gezeigt, kann ein Test nur die Anwesenheit von Fehlern nachweisen, nicht jedoch, dass die Software fehlerfrei ist.

Mit einer Simulation prüft der Lagerbetreiber das LVS vor Inbetriebnahme seines Lagers in einem ersten Abnahmetest. Ohne eine Simulation muss der Abnahmetest auf dem realen Lager erfolgen.

3.2.4. Testfälle

Die Generierung von Testfällen mit den zugehörigen Telegrammen stellt die eigentliche Herausforderung beim Testen von Lagerverwaltungssystem dar. Das Übermitteln der Telegramme über TCP (oder ein anderes Protokoll) ist kein wesentliches Problem.

Blackbox Tests:

Da dieser Arbeit kein LVS zugrunde liegt ist es nicht möglich konkrete Testfälle anzugeben. Stattdessen sollen Ansätze zur Konstruktion von Testfällen aufgezeigt werden. Behandelt man das LVS als Blackbox ist man in der Lage diverse Funktionalitäten zu prüfen:

- Umlagerungen
- Warenannahme mit Avis
- Protokollierung
- Kommunikation mit Transport- oder Staplerleitsystem (soweit im Simulationsmodell abgebildet)

- Kommunikation mit untergelagerten System wie z.B. der FT oder RGBs (soweit im Simulationsmodell abgebildet)
- Einlagermanagement
- Auftragsmanagement
- Bereitstellungsmanagement
- Bestandsführung
- Abwicklung von Warenein- und Ausgangsprozessen
- Verwaltung der Transporteinheiten bzw. Lagereinheiten
- Transportmanagement
- Bereitstellung von Informationen bzw. Übersichten (z.B. über die aktuelle Situation im Lager)
- Wareneingang mit Bestell-Avis
- Chargenverwaltung
- Mandantenfähigkeit
- MHD Verwaltung (Mindesthaltbarkeitsdatum)
- Umlaufkommissionierung nach dem Prinzip Ware zum Mann
- Bypassfunktion
- Cross Docking (Material vom Wareneingang direkt zum Warenausgang)
- Dynamische Lagerplatzverwaltung
- Verdichtung
- Bewegungsjournal

Weitere Funktionen, wie z.B. das Topologiemanagement werden dabei indirekt getestet. Folgende Szenarien sind vorstellbar:

- Inbetrieb- und Wiederinbetriebnahme können simuliert werden (Wiederinbetriebnahme auch mit geändertem Systemstatus[18]) siehe auch Beispiel in Kapitel 3.2.2
- Aufbau und Wiederaufbau der Netzwerkverbindung
- allgemeine Abläufe im Lager (Einlagerungen, Auslagerungen, Umlagerungen, Wareneingang, Warenausgang, etc .)
- defekte Sensoren liefern unsinnige Daten (z.B. Barcode Lesegerät oder Waage)

[18] Beispielsweise könnten sich Standorte von Platten geändert haben, ohne dass dies dem LVS bekannt ist.

- eine Lagereinheit verschwindet (in der Praxis: Palette fällt vom Förderband)
- unsinnige Antworten von untergeordneten Systemen (MFR, Staplerleitsystem, etc.)
- Auslagerung: Am Lagerplatz wird nicht das erwartete Material vorgefunden
    - andere Palette auf dem Lagerplatz, als erwartet
    - keine Palette auf dem Lagerplatz
- Provokation von Staus auf der Fördertechnik, indem Bearbeitungszeiten sehr hoch angesetzt werden
- Lagertopologie im Simulationsmodell verändern (Topologie im LVS stimmt also nicht mit der „realen“ überein)
- simulation eines defekten Geräts (z.B. Regalbediengerät)
- Fehlbestand am Kommissionierplatz (Auftrag kann nicht komplettiert werden)
- Wareneingang wurde nicht Avisiert

Abhängig vom Aufbau des Lagers sind weitere Testszenarien Szenarien denkbar.

Whiteboxtests

Da Whiteboxtests auf den Quellcode des Prüflings aufbauen wird hier nicht weiter darauf eingegangen.

3.2.5. Protokollierung

Die Protokollierung stellt die die Reproduzierbarkeit sicher. In einem Testprotokoll muss zunächst der Startzustand der Umgebung festgehalten werden. Des Weiteren müssen alle empfangenen und gesendeten Telegramme protokolliert werden. Dazu gehört auch die Erfassung von Zeitstempeln, damit die Leistungsfähigkeit des LVS eingeschätzt werden kann. Soweit möglich sind fehlerhafte und unsinnige Telegramme zu dokumentieren. Weißt der Zustand der Simulation auf einen Fehler hin (z.B. Palette auf dem NiO/Nicht in Ordnung Platz, siehe dazu auch die Abbildungen Abbildung 3.1 und Abbildung 3.2), ist auch dies zu protokollieren.

Eine automatische Protokollierung durch die Simulation wäre wünschenswert, da z.B. die Erfassung von Zeitstempeln manuell kaum durchführbar ist.

Abbildung 3.1 *Klärungsband (NiO) bei Lufthansa Technik Hamburg (eigenes Foto)*

Abbildung 3.2 *Schilder über dem Klärungsband, NiO (eigenes Foto)*

3.2.6. Simulationstools

Es gibt Simulationstools, die speziell für die Simulation von logistischen Systemen entwickelt wurden. Man diesen Tools kann man z.B. Lager-, Transport – oder Produktionssysteme simulieren. Eine Auswahl gängiger Simulationstools zeigt Tabelle 3.1.

Hersteller	**Tool**	**Quelle**	**Bemerkungen**
Flexsim	Flexsim 4.3	[FLE-08]	**Siehe** Abbildung 3.3
Tecnomatix (Siemens)	Plant Simulation 8.1	[EMP-08]	Ehem. EM Plant
Enterprise Dynamics	Enterprise Dynamics	[ENT-08]	**Tool, mit dem die Simulation die dieser Arbeit zugrunde liegt erstellt wurde, siehe auch** Abbildung 1.2
SimPlan AG	AutoMod	[AUT-08]	
SimPlan AG	Simul8	[SIM-08]	
ProModel Corporation	ProModel	[PRO-08]	
Lanner	Witness2008	[LAN-08]	
DUALIS GmbH IT Solutions	SpeedSIM	[DUA-08]	
XJ Technologies	Anylogic 6	[XJT-08]	
SDZ GmbH	DOSIMS 3	[SDZ-08]	
SDZ GmbH	SimAL	[SDZ-08]	
SDZ GmbH	SimAL. Trim	[SDZ-08]	
SDZ GmbH	SIMPRO	[SDZ-08]	

Tabelle 3.1 *Simulationstools (Auswahl)*

Viele dieser Tools ermöglichen dem Anwender ein 2D- oder 3D Modell des Lagers aufzubauen und eine Logik zu implementieren, die das Modell steuert. So baut Beispielsweise Flexsim auf der Programmiersprache C++ auf. Andere Simulationstools verfügen über eigene Programmiersprachen. So werden Modelle in Plant Simulation Beispielsweise über eine eigene Sprache namens SimTalk gesteuert. Auch Enterprise Dynamics verfügt mit 4DScript über eine eigene Programmiersprache.

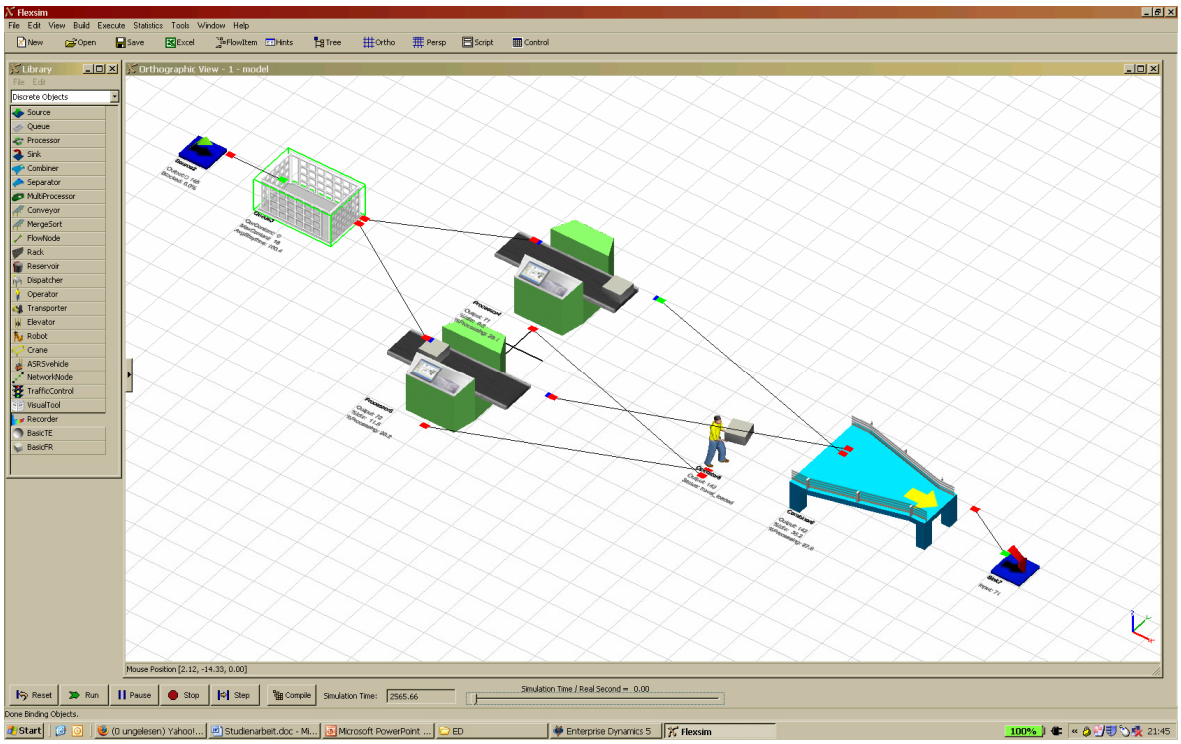


Abbildung 3.3 *Das Simulationstool Flexsim (3D Modell, Screenshot)*

3.3. Weitere Möglichkeiten das LVS zu testen

Das LVS kann mit einem entsprechenden Tool einem Stress- oder Belastungstest ausgesetzt werden um feststellen zu können, wie belastbar das System ist (siehe Kapitel 2.6.6). Auch die Datenbank kann man einem solchen Test unterziehen.

Es bietet sich an, die graphischen Benutzerschnittstellen mit Hilfe der dafür vorgesehenen und in Kapitel 2.6.6 vorgestellten Capture and Replay Tools zu testen.

3.4. Weitere Einsatzmöglichkeiten von Simulationsmodellen zu Testzwecken

Ein Simulationsmodell kann auch eingesetzt werden, um die Ebenen unter dem LVS zu testen. So können (je nach Lager) z.B. der MFR, das Staplerleitsystem oder auch eine SPS getestet werden [NOC-08, S. 2]. Regressionstest bieten sich an, wenn nur eine dieser Komponenten ausgetauscht werden soll. Das Verhalten des neuen Produkts muss denselben Spezifikationen genügen, wie das alte. Andernfalls müssen auch andere Systemkomponenten ausgetauscht oder angepasst werden.

3.5. Test einer Simulation

3.5.1. Testansätze um eine Simulation zu testen

Um eine Simulation zu testen muss diese zunächst gestartet werden. Das gegebene Simulationsmodell kann mit Hilfe des Safeline Standards (siehe 2.2) gesteuert werden. Dies trifft sicher nicht immer zu, da auch andere Kommunikationsstandards existieren, als Basis wird dabei auch nicht immer TCP verwendet. Die einfachste Art und Weise ein TCP-Telegramm an die Simulation zu übermitteln stellt daher das Programm telnet dar, das unverschlüsselte TCP Pakete versendet und empfängt. Diese Methode ist sehr primitiv und bietet keinerlei Automatisierung.

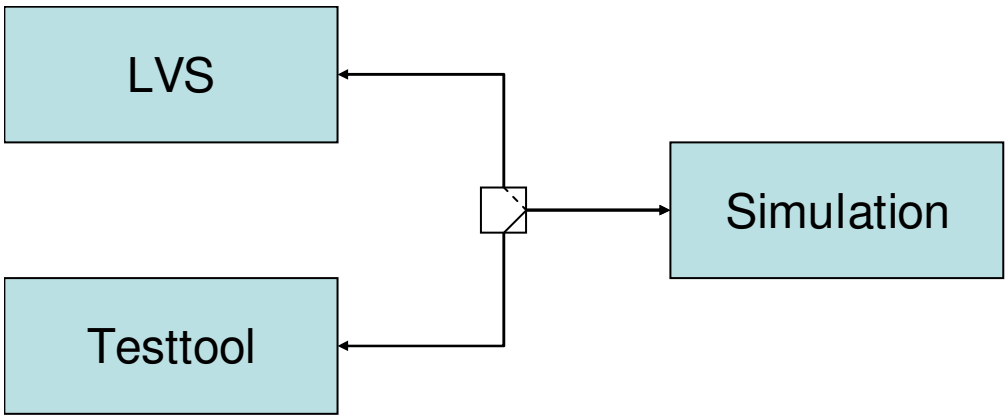


Abbildung 3.4 *Testsituation: Die Simulation kommuniziert mit dem Testtool statt mit dem LVS (eigene Darstellung)*

Eine weitere Möglichkeit ist der Einsatz eines vollwertigen LVS, dass den Safeline Standard unterstützt. Diese Variante scheidet während der Entwicklung aus mehreren Gründen aus:

1. Das LVS muss an das Lager bzw. die Simulation angepasst werden und steht damit zu Beginn der Entwicklung noch nicht zur Verfügung (Anpassung des LVS und Entwicklung der Simulation finden parallel statt, dies nennt man auch concurrent engineering[19])
2. Ein LVS ist sehr teuer. Soweit nicht der LVS Hersteller die Entwicklung der Simulation übernommen hat, wird er nur ungern seine Software kostenneutral zur Verfügung stellen.

[19] Nebenläufige oder parallele Entwicklung

3. Ein Lagerverwaltungssystem ist kein Testtool und bietet keinerlei Möglichkeiten der automatisierten Prüfung

Der Einsatz einer Testsoftware bietet sich daher an. Diese muss Telegramme an die Simulation senden und ihre Antworten empfangen können. Will man den Test automatisieren, übergibt man dem Tool die zu erwartenden Antworten. So ist das Programm in der Lage zu prüfen, ob die von der Simulation gelieferten Telegramme den Erwartungen entsprechen. Die Simulation kann nicht zwischen Testtool und LVS unterscheiden. Die Testsituation ist in Abbildung 3.4 dargestellt.

3.5.2. Software zum Testen von Simulationen

3.5.2.1. Ausgangssituation

Um eine Simulation zu testen ist es nötig Telegramme an diese zu senden und die Antworten wenn möglich automatisiert auszuwerten.

3.5.2.2. Testfälle

Die Simulation soll spezifische Testfälle beherrschen (Beispiele: siehe Kapitel 3.2.2). Diese Testfälle lassen sich vom Prüfling LVS auf den Prüfling Simulation übertragen. Dabei müssen mehrere Punkte getestet werden:

1. Korrektheit der Antworttelegramme
2. Verhalten der Simulation
3. Protokollierung

Die Überprüfung der Telegramme stellt die Funktionalität der Simulation gegenüber dem LVS sicher. Zusätzlich muss die Simulation sich entsprechend den Spezifikationen verhalten.

Beispiel:

Ein unsinniges Telegramm (z.B. die Einlagerung in eine nicht existente Gasse) sollte dazu führen, dass die Lagereinheit am NiO (Nicht in Ordnung) Platz ankommt. Wird sie stattdessen in eine andere Gasse eingelagert fällt der entsprechende Fehler nicht auf und kann auch nicht protokolliert werden.

Das Verhalten der Simulation kann nicht ohne weiteres automatisch geprüft werden, da ihr Zustand dazu automatisch erfasst werden müsste. Hier wird im Allgemeinen der manuelle Test nötig sein.

Der Test der Protokollierung ist vor allem dann entscheidend wichtig, wenn im Test des LVS Fehler primär über das Protokoll gesucht werden und manuelle Prüfung nicht oder nur wenig stattfindet. Wird ein Fehler im Protokoll nicht aufgeführt, kann er dann nicht erkannt und behoben werden.

3.5.2.3. Software auf dem freien Markt

Es existieren viele kostenfreie und kostenpflichtige Programme, die das Testen von Software in vielerlei Hinsicht unterstützen können. Eine vollständige Übersicht würde jedoch den Rahmen dieser Arbeit sprengen. Daher werden hier nur einige Beispiele aufgeführt:

Beispiele

- TCP Test Tool (http://www.simplecomtools.com)
- Network Traffic Generator and Monitor (http://www.pbsoftware.org/id17.html)
- TrafficEmulator (http://www.nsauditor.com/docs/html/tools/Traffic%20Emulator.htm)

Das TCP Test Tool sendet und empfängt TCP Pakete. Man kann so auf relativ einfache Art und Weise Pakete an die Simulation senden und Pakete von ihr empfangen. Es bietet allerdings keine Möglichkeit Pakete zu speichern, oder Pakete auf Korrektheit zu prüfen. Das Tool ist in
Abbildung 3.5 gezeigt. Im Tool sind ein Client und ein Server verfügbar, beide können sowohl Pakete senden, als auch empfangen. Der Client baut eine Verbindung zu einem beliebigen Server auf, der Server nimmt bis zu 250 Verbindungen an. In einer Statusleiste werden Statistische Daten (z.B. gesendete Bytes, empfangene Bytes) angezeigt. Es ist nicht möglich zwei Instanzen des Programms zu starten. Dies stellt einen erheblichen Nachteil dar. Die gegebene Simulation erwartet zwei Server, mit denen sie sich verbinden kann. Ein Test der Simulation des Tchibo Lagers ist mit diesem Tool daher nur eingeschränkt möglich.

TCP Test Tool ist Bestandteil einer Serie von Testtools, die SimpleComTools genannt wird. Unter anderem existiert in dieser Serie auch ein Tool, mit dem UDP Pakete versendet werden können. Diese Tools sind kostenfrei erhältlich.

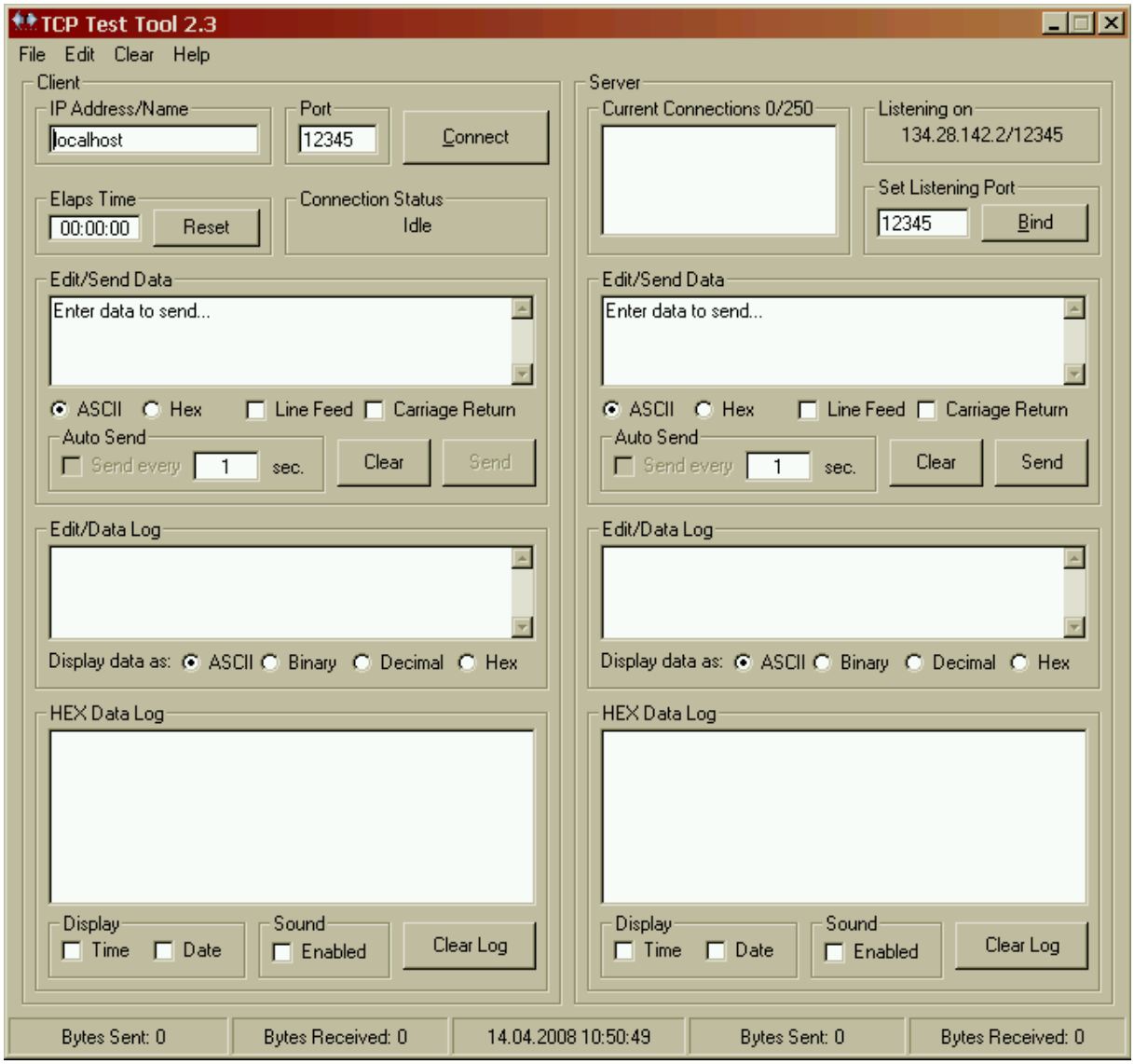


Abbildung 3.5 *Das TCP Test Tool (Screenshot)*

Der Network Traffic Generator and Monitor ist dagegen Shareware und kostet 350$. Er ist in der Lage mehrere Verbindungen zu halten und zu Monitoren. Dabei können unterschiedliche Protokolle zum Einsatz kommen.

Der TrafficEmulator unterstützt den Testingenieur bei der Durchführung von Last- und Stresstests.

3.5.2.4. Ansätze zur Implementierung eines Testtools

Es ergeben sich verschiedene Ansätze ein Tool zum Testen eines Simulationsmodells umzusetzen. Ein solches Tool sollte mehr Funktionalität bieten, als telnet und mit relativ geringem Aufwand implementiert werden. Grundsätzlich lassen sich mehrere wünschenswerte Funktionalitäten identifizieren:

- Senden und Empfangen von Telegrammen
- Speichern von Telegrammen
- Speichern von erwarteten Antworten
- Automatische Auswertung der Antworten
- Automatisches Prüfen der Reaktion des Simulationsmodells
- Erzeugung der Testfälle

Telnet ist in der Lage den ersten Punkt abzudecken (dazu gehört auch der Aufbau der Verbindung). Das Speichern von Telegrammen beinhaltet sowohl Telegramme, die gesendet werden sollen, als auch Telegramme, die empfangen wurden. Werden die erwarteten Antworten vorgegeben, hat man die Möglichkeit der Prüfung. Telegramme oder ganze Testfälle können in einer Datenbank abgelegt werden.

Die letzten beiden Aspekte (automatisches Prüfen der Reaktion des Simulationsmodells und Erzeugung der Testfälle) sind nur bedingt mit vertretbarem Aufwand umsetzbar. Ein Zugriff auf den Zustand des Simulationsmodells ist nicht ohne weiteres möglich. Zudem gibt es für die Beschreibung des Zustandes keine Standards. Das Erzeugen von sinnvollen Testfällen, in denen zeitlich geordnet Telegramme in Abhängigkeit der Antworten gesendet werden, wird am ehesten von einem LVS geleistet, da es die Abhängigkeit von den Antworten nötig macht, die Telegramme zur Laufzeit zu erzeugen. Dies stellt einen erheblichen Aufwand dar. Lässt man jedoch die Abhängigkeit von den Antworttelegrammen außen vor besteht die Möglichkeit vor Testbeginn eine Reihe Telegramme und erwarteter Antworten zu erzeugen (z.B. eine Reihe Auslagerauftragstelegramme, siehe Kapitel 2.2.2.3).

Die Generierung von zufälligen Testfällen ist kaum möglich, da die Telegramme den Vorgaben aus dem Kapitel 2.2 entsprechen müssen. Auch die erwarteten

Antworttelegramme können nicht automatisch generiert werden. Hier sind Vorgaben durch den Tester notwendig.

3.5.2.5. Vorschlag zum Vorgehen

Der Aufbau einer TCP Verbindung zur Simulation ist relativ einfach umsetzbar und auch das Senden und Empfangen von TCP-Paketen stellt kein großes Problem dar. Diese Funktionalitäten können in einem einfachen Webinterface umgesetzt werden.

Das Speichern von Telegrammen in einer Datenbank stellt keine besonderen Anforderungen und auch der Vergleich von empfangenen mit erwarteten Telegrammen ist relativ einfach und schnell realisierbar. Löschen, ändern und erstellen von Telegrammen ist unproblematisch über ein Webinterface lösbar.

Die Generierung von Testdaten stellt dagegen relativ große Ansprüche, da sie den Spezifikationen des Safeline Standards entsprechen und die Simulation sinnvoll steuern müssen. Da die Implementierung von Simulationsmodellen auch durch „Nichtinformatiker“ geschieht, wäre es von Vorteil die Daten für die Testfallgenerierung möglichst intuitiv einzugeben. Eine Vorgabe dieser Arbeit war die Bedienung des Tools über ein Webinterface. Daher bietet es sich an, ein Interface zu erstellen, das dem Administrationsinterface eines LVS nachempfunden ist. Es müssten hier also Aufträge und Bestände, Bewegungen, Avis und Lagereinheiten abgebildet und in einer Datenbank persistent gespeichert werden. Ein Testdatengenerator müsste dann, basierend auf den Daten aus der Datenbank die Testfälle generieren.

3.5.2.6. Ausblick

Das Testtool kann nicht nur zum Testen einer Simulation eingesetzt werden. Wenn entsprechende Telegramme und Testfälle hinterlegt sind kann man jede TCP basierte Software prüfen. Eine Erweiterung des Tools, um andere Protokolle könnte den Anwendungsbereich ausweiten. So könnten beispielsweise auch Materialflussrechner, „Speicherprogrammierbare Steuerungen“ (SPS) oder Staplerleitsysteme geprüft werden.

4. Analyse

4.1. Use cases Test

Der Nutzer will Telegramme erstellen, in der Datenbank speichern, aus der Datenbank laden und an die Simulation senden. Des Weiteren sollen Testfälle erstellt werden, die aus Telegrammen bestehen, die gesendet bzw. als Antwort erwartet werden. Außerdem ist zu prüfen, ob eine Antwort auch erwartet wird. Dazu wird sie mit den erwarteten Antworten in der Datenbank verglichen. Die Anwendungsfälle sind im Use Case in
Abbildung 4.1 dargestellt.

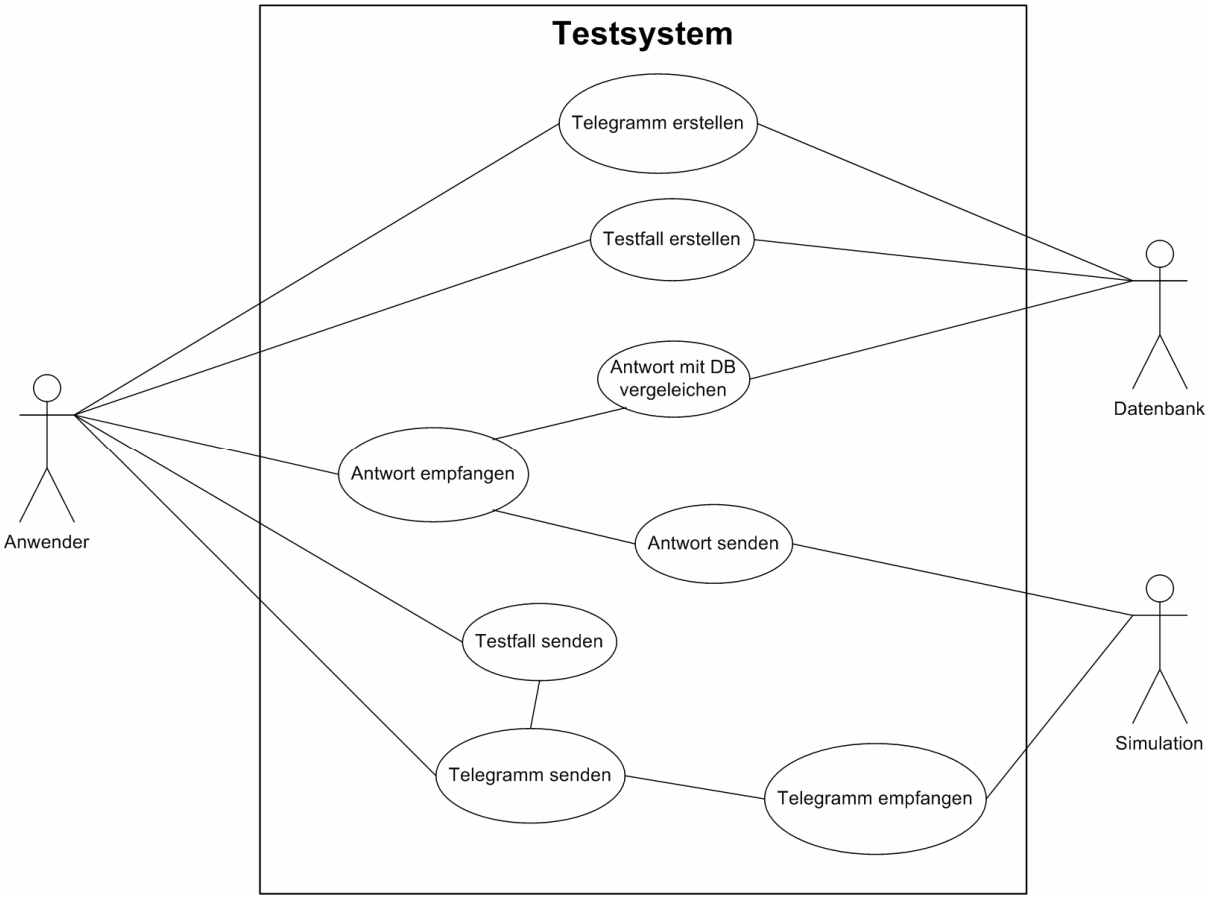


Abbildung 4.1 *Use Case Testsystem: Telegramme und Testfälle (eigene Darstellung)*

4.1.1. Telegramm erstellen

Ein Benutzer erstellt Telegramme. Er gibt dazu einen Telegrammtext und ein Beschreibung an. In der Beschreibung wird festgehalten, welche Telegrammart vorliegt (z.B. Auslagertelegramm)

4.1.2. Testfall erstellen

Ein Benutzer erstellt einen Testfall. Dazu gibt er Namen und Beschreibung, sowie mehrere Telegramme an. Er vermerkt, ob die Telegramme vom Testtool gesendet oder empfangen werden. Dieser Anwendungsfall impliziert den use case „Telegramm zu Testfall hinzufügen“.

4.1.3. Telegramm senden

Ein Telegramm wird an eine Simulation gesendet. Dabei wird der Telegrammtext übermittelt und der Zeitpunkt zu dem das Telegramm empfangen wurde protokolliert. Das kann z.B. eine Auslagerung oder die Freigabe einer Einlagerung sein. Diese Vorgänge lösen später Antworttelegramme aus.

4.1.4. Telegramm empfangen

Die Simulation empfängt ein Telegramm und löst die nötigen Vorgänge im Simulationsmodell aus.

4.1.5. Testfall senden (Test starten)

Wird ein Test gestartet, werden alle im aktiven Testfall enthaltenen Telegramme gesendet, die vom Typ ‚S’ sind. Antworten werden darauf geprüft, ob im Testfall ein Telegramm vorliegt, dass den Typ ‚E’, sowie den gleichen Telegrammtext hat.

4.1.6. Antwort empfangen

Eine Antwort der Simulation geht ein. Nach dem Empfang des Telegramms wird geprüft, ob es erwartet wurde. Der Empfang eines Telegramms wird protokolliert.

4.1.7. Antwort prüfen (Antwort mit DB vergleichen)

Ein Telegramm von der Simulation wird erwartet, wenn es im Testfall vorgegeben wurde. Das Ergebnis dieser Prüfung muss protokolliert werden.

4.2. Use Cases Datengenerierung

Um Telegramme und Testfälle generieren zu können werden Eingangsdaten benötigt. Um auf einen Wareneingang antworten zu können bedient sich ein Lagerverwaltungssystem avisierten Aufträgen. Die Möglichen use cases für diese Daten sind in Abbildung 4.2 gezeigt.

Die use cases für andere Daten sehen diesem sehr ähnlich und werden daher nicht aufgeführt.

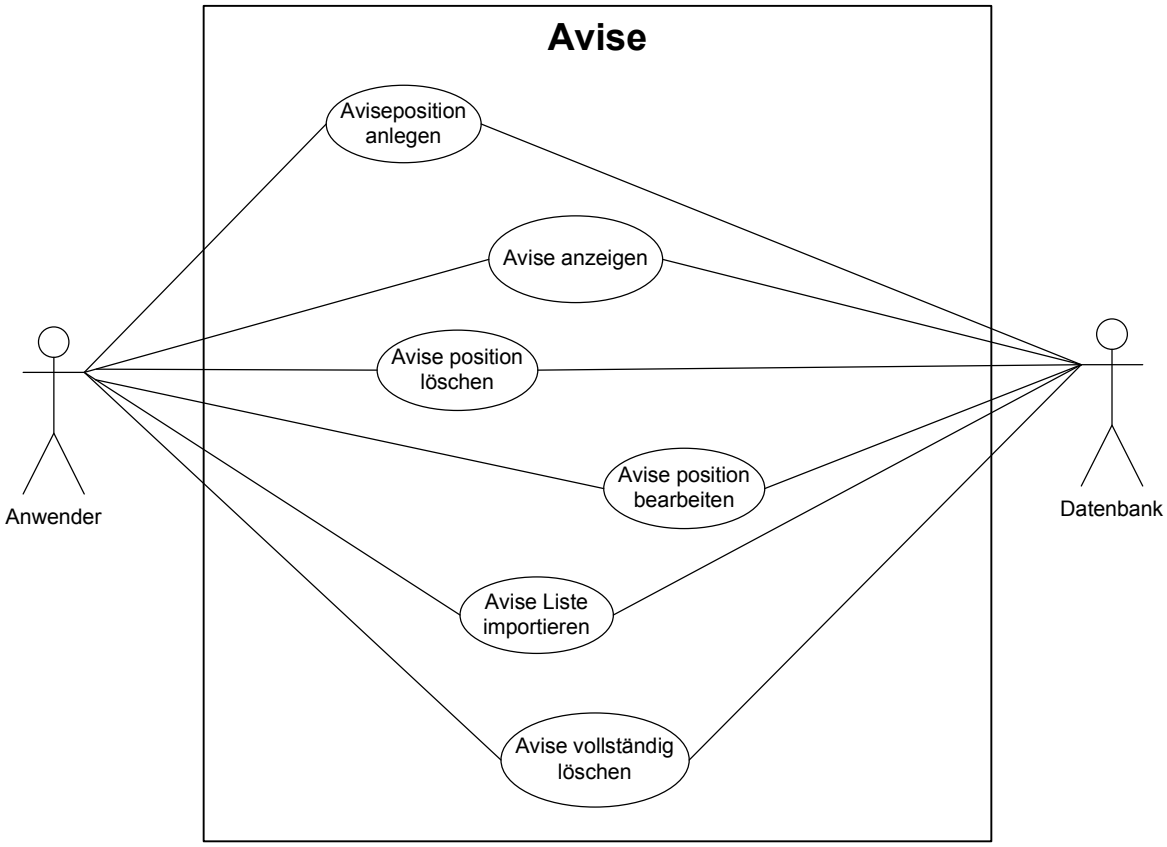


Abbildung 4.2 *Use cases für Avisedaten (eigene Darstellung)*

4.2.1. Aviseposition anlegen

Die Position einer Avise beschreibt zu welchem Auftrag die Position gehört und wie viele Teile von welchem Artikel erwartet werden.

4.2.2. Avise anzeigen

Zeigt eine tabellarische Übersicht über alle avisierten Positionen in der Datenbank. Jede Position soll aus der Liste heraus editiert oder gelöscht werden können.

4.2.3. Aviseposition löschen

Entfernung der Position aus der Liste der avisierten Positionen

4.2.4. Aviseposition bearbeiten

Zeigt ein Formular an, das die aktuellen Werte einer avisierten Position enthält, die Möglichkeit bietet diese zu ändern und anschließend zu speichern.

4.2.5. Avise Liste importieren

Es soll die Möglichkeit bestehen eine Liste von avisierten Positionen zu importieren. Dazu wird der Standarddialog des Betriebssystems zum Öffnen von Dateien angezeigt und die ausgewählt Liste importiert. Die Liste soll mit einer Tabellenkalkulation erstellt werden können. Die Spezifikationen sind entsprechend zu wählen

4.2.6. Avise vollständig löschen

In diesem use case werden alle vorhandenen Positionen in der Avise Liste gelöscht.

5. Entwürfe

5.1. Übersicht

Soll ein Werkzeug Telegramme mit einer Simulation austauschen, um diese zu testen muss das Tool über ein Modul zum senden und empfangen von Telegrammen (über TCP) verfügen. Telegramme und Testfälle können um Tool als eigene Klassen abgebildet werden.

Ein Interface zur Generierung für Testdaten könnte Klassen für die einzelnen Daten (siehe Kapitel) nutzen.

5.2. Entwurf für eine Klasse Testtool

Die Klasse Testtool ist zuständig für die Interaktion mit der Simulation und das prüfen von Telegrammen. Die Prüfung der Telegramme erfolgt durch einen Abgleich mit den im Testfall hinterlegten, erwarteten Telegrammen. Die Klasse soll einen Testfall laden und abarbeiten.

Dazu müssen zunächst alle Telegramme geladen werden, die zum Testfall gehören. Die Telegramme werden entweder erwartet oder sollen gesendet werden, letztere werden an die Simulation übermittelt. Die Übermittlung wird zusammen mit einem Zeitstempel protokolliert. Die Antworten werden mit den erwarteten Telegrammen verglichen und ebenso protokolliert. Wird ein nicht erwartetes Telegramm empfangen, wird dies entsprechend im Protokoll vermerkt.

5.3. Entwurf für die Telegramm Objekte

Telegramme enthalten einen String Telegrammtext und einen String Beschreibung. Sie werden in der Datenbank gespeichert und aus ihr geladen. Ein Telegramm wird entweder gesendet oder empfangen. Wenn es empfangen wird, muss geprüft werden, ob es erwartet wurde.

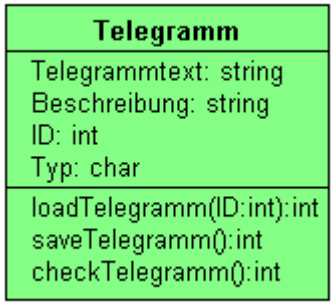


Abbildung 5.1 *Die Klasse Telegramm (eigene Darstellung)*

Jedes Telegramm kann durch seine ID eindeutig identifiziert werden. Testfälle in Telegrammen haben einen Typ. Wenn ein Telegramm erwartet wird gilt Typ = ‚E'. Soll das Telegramm gesendet werden ist der Typ = ‚S'.

5.4. Entwurf für ein Objekt Testfall

Ein Testfall besitzt die folgenden Eigenschaften

- ID
- Name
- Beschreibung
- Telegramme

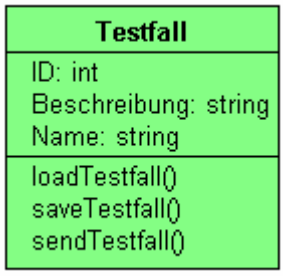


Abbildung 5.2 *Die Klasse Testfall (eigene Darstellung)*

Bei den Telegrammen müssen die Typen entweder auf ‚S' oder auf ‚E' gesetzt werden. Eine Relation Telegramme_Testfall bestimmt, welche Telegramme in welchem Testfall (auch mehrfach) enthalten sind und ob sie erwartet oder gesendet werden.

5.5. Entwürfe für ein LVS Interface zur Testfallgenerierung

5.5.1. Zu erzeugende Telegramme (senden)

Es sollte die Möglichkeit bestehen alle Telegrammarten, die vom LVS an den MFR gesendet werden zu erzeugen. Diese sind in Tabelle 2.5 aufgeführt. Folgende Situationen treten auf:

- Auslagerauftrag erzeugen
- Auftrag wieder stornieren
- Folgeauftrag erteilen

} Identischer Aufbau

- An/Abmeldung von Leerpaletten
- Artikel An- und Abmeldung
- Freigabe Vortakten WE Bahn
- Freigabe Einlagerung WE Bahn
- Freigabe Verladung WA Bahn

5.5.2. Zu erzeugende Telegramme (empfangen)

5.5.3. Benötigte Daten

Um Testfälle Generieren zu könnte man folgende Daten nutzen:

- Bestandsdaten
  - Artikel ID
  - Menge
  - Lagerhilfsmittel ID (Paletten ID oder Boxen ID)
- Lagerhilfsmittel (LHM)
  - LHM ID
  - Standort
  - Quelle
  - Ziel
- Bewegungsaufträge
  - Auftrags ID
  - Auftragsart (Auslagerauftrag, Einlagerauftrag …)

  - Priorität
  - Ziel
  - Auftragspositionen
  - Artikel ID
  - Menge
  - LE
- Kundenaufträge
  - Kundenauftrags ID
  - Kundenname
  - Artikel ID
  - Menge
  - Zugehörige(r) Bewegungsauftrag
- Avisedaten (Welche Artikel werden erwartet)
  - Auftrags ID
  - Artikel ID
  - Menge

Auch Artikelstammdaten könnten herangezogen werden. Aus den Avisedaten könnten Antworttelegramme auf eine Anfrage aus dem Wareneingang generiert werden. Bei einer Anlieferung fragt der Wareneingang nach, ob die angelieferte Ware in gegebener Menge erwartet wird. Anschließend ließen sich Einlagertelegramme generieren; dazu muss nur ein Ziel vorgegeben werden. Aus Kundenaufträgen ließen sich Auslageraufträge und die dazu gehörigen Telegramme generieren. Ein Bewegungsauftrag gibt die Möglichkeit zu prüfen, ob ein Telegramm von der Simulation erzeugt wird, sobald die LE an ihrem Ziel ankommt.

5.6. Aktivitätsdiagramme

Alle Daten (also Telegrammen, Testfällen, Avise usw.) müssen über ein Webinterface eingegeben und anschließend in der Datenbank abgespeichert werden. Die auszuführenden Maßnahmen sind im Aktivitätsdiagramm in Abbildung 5.3 gezeigt.

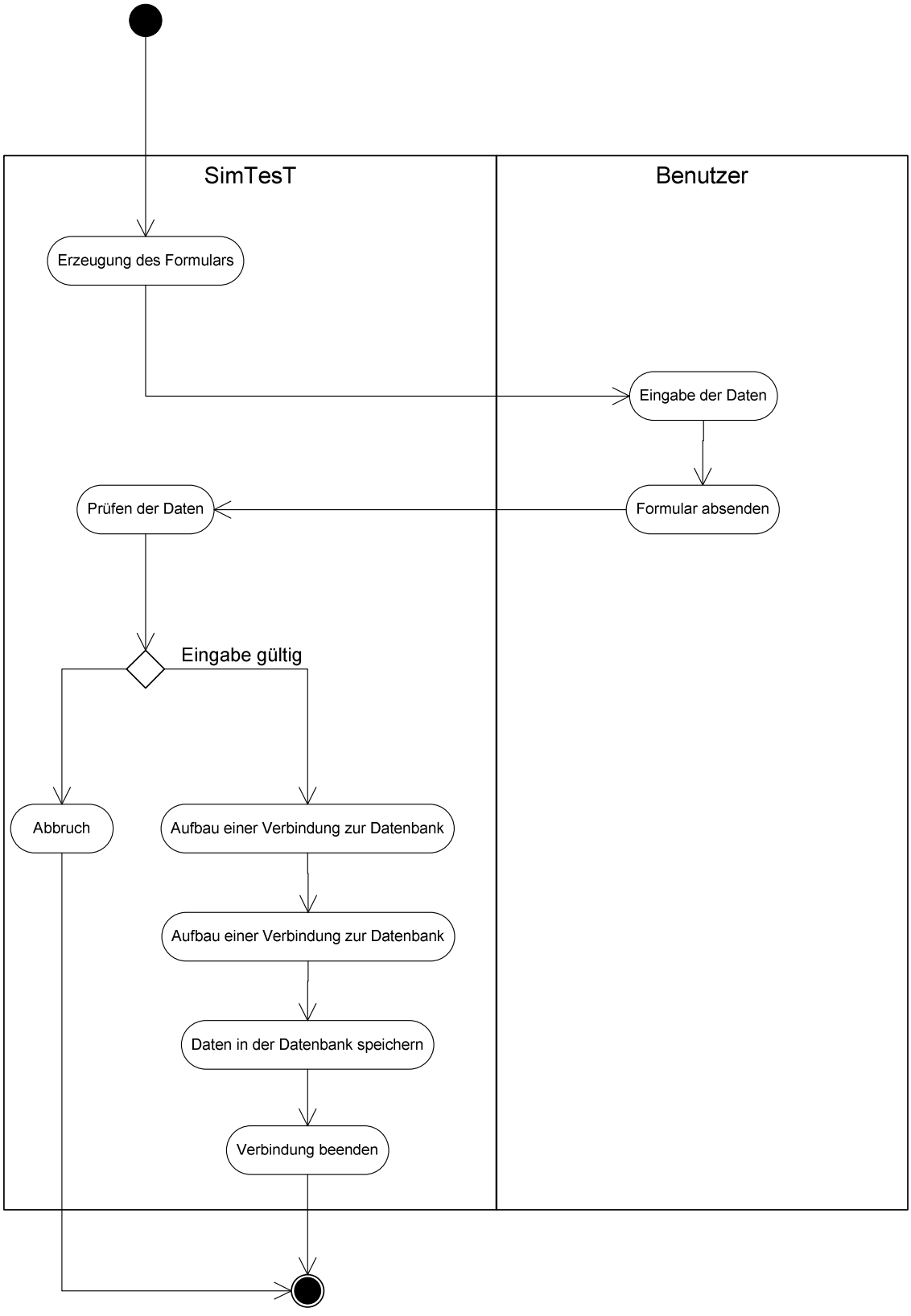


Abbildung 5.3 *Aktivitätsdiagramm: Daten in der Datenbank speichern (eigene Darstellung)*

5.7. Datenbankentwurf

Folgende Tabellen sind für die Speicherung von Telegrammen und Testfällen nötig:

- Telegramme
- Testfälle
- Telegramme_Testfälle (Zuordnung: welche Telegramme sind in welchem Testfall enthalten)

Die Tabellen stellen sich wie folgt dar:

Telegramme

ID	Telegrammtext	Beschreibung
1	00MFLVHNO2	Auslagertelegramm
2	00LVMF0815	Einlagertelegramm

Testfälle

ID	Name	Beschreibung
1	Auslagerungen	Mehrere Auslagerungen aus Regal 1
2	Einlagerungen	Einlagerungen in Regal 2
…	…	…

Telegramme_Testfälle

Telegramm	Testfall	Typ
1	1	E
1	2	S
2	2	S
3	2	E

Das Feld Typ enthält entweder ein ‚S' für senden (das Telegramm soll also gesendet werden) oder ein ‚E' für empfangen (das Telegramm wird also erwartet).

6. Eingesetzte Software

6.1. Ausgangssituation

Wie bereits in 1.2 angeführt soll mit dem Testtool gezeigt werden, dass Fehler in Lagerverwaltungssystemen mit Hilfe einer Simulation aufgezeigt werden können. Um das LVS zu implementieren werden unterschiedliche Softwarekomponenten eingesetzt.

Das LVS benötigt eine Datenbank, um Auftrags-, Bestands, und Avisedaten sowie die Topologie des Lagers persistent zu speichern. Zusätzlich soll das LVS über ein Webinterface verfügen, das grundlegende administrative Aufgaben unterstützt. Dazu werden ein Webserver sowie die Datenbank benötigt. Um den Code vom Design der Webseite trennen zu können soll eine Templateengine eingesetzt (siehe 6.4) werden. Da eine Weiterentwicklung der Software wahrscheinlich ist sollen die einzelnen Komponenten dieser nicht entgegenstehen.

Da Simulationssoftware im Allgemeinen nur unter Windows verfügbar ist wird auch das LVS in dieser Umgebung implementiert. Sowohl LVS, als auch das Webinterface sollten aber auch auf anderen Plattformen lauffähig sein, vorausgesetzt die eingesetzten Softwarekomponenten stehen zur Verfügung.

6.2. Programmiersprache

Da die Administration des LVS über ein webbasiertes Interface erfolgen soll und gleichzeitig zu erwarten ist, dass die Software im Laufe der Zeit angepasst werden muss wurde die Entscheidung getroffen, für Webinterface und LVS die gleiche Programmiersprache nutzen, dies erleichtert die Einarbeitung in das Projekt. An die Programmiersprache werden folgende Anforderungen gestellt:

- TCP/IP Socket Unterstützung
- Nutzbar sowohl für die Webanwendung als auch für den Server
- Datenbankunterstützung
- Die Nutzung von Templates soll unterstützt werden
- Eine spätere Weiterentwicklung soll unterstützt werden

Drei für Webanwendungen häufig eingesetzte Open Source Skriptsprachen sind:

1. PHP
2. Perl
3. Python

Weitere hier nicht betrachtete, zum Teil proprietäre Alternativen sind:

1. Active Server Pages (kostenpflichtig, nicht auf apache verfügbar) in Kombination mit VBSkript, Perl oder JScript
2. JSP (JavaServer Pages)
3. SSI (Server Side Includes, reine WWW Skript-Sprache, sehr rudimentär)

Sprache	Vorteile	Nachteile
PHP	• weit verbreitet • Objektorientierung möglich • plattformunabhängig • Erfahrung vorhanden	• TCP/IP Unterstützung zu Anfang der Studienarbeit nur als testing (die Stabilität ist nicht gewährleistet, Packet könnte wieder entfernt werden) • Kernkompetenz ist die Webanwendung
Perl	• plattformunabhäng • relativ weit verbreitet • Dokumentation im Quelltext (POD) • objektorientiert • there is more than one way to do it“, kurz TIMTOWTDI (Tim Today)	• There is more than one way to do it“, kurz TIMTOWTDI (Tim Today), kann zu Write-only Programmen führen und damit unter Umständen zu sehr unleserlichem Code. Dies würde eine Weiterentwicklung schwierig gestalten. • eignet sich eher für kleine, „schnelle“ Programme
Python	• sichert einen Übersichtlichen, lesbaren Code (Wichtig für Weiterentwicklung) • objektorientiert • olattformunabhäng • Dokumentation im Quelltext • eignet sich gut für große Projekte (ist im Einsatz z.B. bei Google oder Yahoo!)	• Die am wenigsten verbreitete der hier aufgeführten Sprachen

Tabelle 6.1 *Vor- und Nachteile der Skriptsprachen PHP, Perl und Python*

Prinzipbedingt ist keine der aufgeführten Programmiersprachen sehr performant. Da die zu entwickelnde Software nur für Demonstrations- und Testzwecke eingesetzt werden soll stellt dies kein Problem dar.

```
@P=split//,".URRUU\c8R";@d=split//,"\nrekcah xinU / lreP rehtona tsuJ";sub p{
@p{„r$p","u$p"}=(P,P);pipe"r$p","u$p";++$p;($q*=2)+=$f=!fork;map{$P=$P[$f^ord
($p{$_})&6];$p{$_}=/ ^$P/ix?$P:close$_}keys%p}p;p;p;p;map{$p{$_}=~/^[P.]/&&
close$_}%p;wait until$?;map{/^r/&&<$_>}%p;$_=$d[$q];sleep rand(2)if/\S/;print
```

Code 6.1 *Beispiel für „write only" Code in Perl [WIK-08]*

Die Stärken der Sprache PHP liegen in den Webanwendungen, für die zahlreiche Funktionen in Frameworks wie beispielsweise PEAR (PHP Extension and Application Repository) zur Verfügung stehen. Andere Anwendungen können zwar umgesetzt werden, im Allgemeinen wird PHP hier aber selten eingesetzt. Aufgrund der, anfänglich von Seiten der Entwickler als nicht stabil eingestuften Socket Funktionalität schied PHP endgültig aus. Inzwischen ist diese aber fest in PHP integriert.

Da zu erwarten ist, dass das hier entwickelte LVS in Zukunft weiterentwickelt und modifiziert wird, schied Perl aus. Der Beispiel Code 6.1 stellt den worst-case eines Perl Programms dar (das Beispiel zeigt ein lauffähiges Perl Programm, bei dem versucht wurde im Rahmen eines sogenannten Obfuscation Wettbewerbs die Funktionalität des Programms zu verbergen). Die Fähigkeit von Perl unterschiedlich formulierten Code zu verstehen (siehe auch BeispielCode 6.2) stellt einen beachtlichen Vorteil dar, wenn kleine Programme realisiert werden sollen. Für große Projekte, die unter Umständen von anderen Personen weiterentwickelt werden sollen ist diese Eigenschaft jedoch eher hinderlich, da sich die unterschiedlichen Entwickler nicht immer gleiche Formulierungen verwenden werden. Der Code wird so unübersichtlich.

```
For (int I = 0; I < 11; i++)
 {
  …
 }

for I = 0 to 10
 {
  …
 }
```

Code 6.2 *Beispiel für Codevarianten in Perl*

Python erleichtert dem Programmierer eine saubere Programmierung, beispielsweise werden Blöcke nicht in Klammern gefasst, sondern durch Einrückungen gekennzeichnet. Dies erhöht die Lesbarkeit eines Programmes. Python erfüllt alle Anforderungen, die gestellt wurden, daher fiel die Wahl auf diese Sprache. Python liegt zur Zeit in Version 2.5 vor und ist eine interpretierte Sprache, für die Interpreter auf allen gängigen Plattformen zur Verfügung stehen.

Die Eignung von Python für eine schnelle Softwareentwicklung wurde in der Diplomarbeit von Ingo Linkweiler [LIN-02] untersucht. Demnach ist die Entwicklung von Programmen in Python nicht nur schneller, sondern auch weniger fehleranfällig, als in anderen Sprachen (Perl, Java, C/C++).

6.3. Webserver und Datenbank

Als Webserver wird Apache in der Version 2.2, als Datenbank MySQL in Version 5.0.25 eingesetzt. Beide sind sehr weit verbreitete Anwendungen (Apache wird laut [NET-08] auf 60% aller Domains eingesetzt) die häufig in Kombination benutzt werden. Dies erleichtert eine spätere Weiterentwicklung.

6.4. Templates zur Darstellung des Administrationsinterfaces

Templates werden genutzt, um Code und Design einer Webseite zu trennen. Das Programm selber stellt nur noch die Variablen zur Verfügung und ruft das Template bzw. die Templateengine auf. Diese fügt dann die entsprechenden Werte in eine Vorlage ein. In

dieser Studienarbeit wird Cheetah eingesetzt, eine relativ weit verbreitete Template Engine (http://www.cheetahtemplate.org/). Sie findet unter anderem auf folgenden Webseiten verwendung:

- Dell
- Ebay & PayPal
- Nasdaq
- Sony
- CNN
- Harvard University
- University of British Columbia
- University of St. Andrews (Scotland)
- Gentoo Linux

```
<html>
    <head><title>$title</title></head>
    <body>
      <table>
        #if $zeigeClient
        #for $client in $clients
        <tr>
          <td>$client.surname, $client.firstname</td>
          <td><a href="mailto:$client.email">$client.email</a></td>
        </tr>
        #end for
        #end if
      </table>
    </body>
  </html>
```

Code 6.3 *Beispiel für ein Cheetah Template*

Wie der
Code 6.3 zeigt ist es möglich in den Templates auf Objekte zuzugreifen und eine beschränkte Entscheidungslogik zu nutzen. Das Beispiel belegt die vielfältigen Einsatzmöglichkeiten von Templates.

6.5. Entwicklungsumgebung

Die Entwicklungsumgebung SPE unterstützt die Entwicklung von Python Programmen und ist selber auch in Python implementiert.

Funktionalitäten von SPE (Auszug):

- Syntax Highlighting
- Syntax Überprüfung
- Ausklappbare Blöcke
- Darstellung des Programms in UML
- Automatische Erstellung von Dokumentation
- Debugging Unterstützung
- GUI Design Unterstützung
- Shell
- Tabbing

6.6. Softwareübersicht und Versionen

Tabelle 6.2 führt die in dieser Arbeit eingesetzten Softwarekomponenten auf. MySQLdb, das den zugriff auf MySQL unter Python ermöglicht wird für Windows nicht durch den Programmierer zur Verfügung gestellt, steht aber als binary im Forum des Projektes auf Sourceforge.org zur Verfügung.

Software	Version	Funktion/Bemerkung
Python	2.5	Programmiersprache
SPE	0.8.2.a	Stanis Python Editor
Apache	2.2	Webserver
MySQL	5.0.25	Datenbank
wxPython	2.6.3.3	User Interface Library für SPE
MySQLdb	1.2.1_p2	MySQL support für Python, (Windows build von William Stearns)
Cheetah	2.0rc7	Template Engine für Python

Tabelle 6.2 *In der Studienarbeit eingesetzte Software*

7. Fazit

7.1. Umgesetzte Funktionalität

Insgesamt wurden 28 Klassen implementiert, sowie mehr als 1800 Zeilen Code und Kommentar geschrieben, zusätzlich wurden 28 Templates angelegt. Mit der Software SimulationsTestTool (SimTesT) wurden folgende Funktionalitäten umgesetzt:

- Telegramme
  - Anzeigen von Telgrammen
  - Erstellen von Telegrammen
  - Ändern von Telegrammen
  - Löschen von Telegrammen
  - Prüfen von Telegrammen
  - Alle Teleramme löschen
- Testfälle
  - Anzeigen von Testfällen
  - Anzeigen von Telegrammen in Testfällen
  - Erstellen von Testfällen
  - Hinzufügen von Telegrammen zu einem Testfall (Relation mit Typ ‚S' oder ‚E')
  - Entfernen von Telegrammen aus einem Testfall
  - Löschen von Testfällen
  - Alle Testfälle löschen
- Avise
  - Anzeigen der Avise
  - Hinzufügen von avisierten Positionen
  - Ändern von avisierten Positionen
  - Löschen einzelner avisierter Positionen
  - Löschen eines avisierten Auftrags
  - Löschen aller avisierten Positionen
  - CSV Dateien importieren
- Bewegungsaufträge
  - Anzeigen von Bewegungsaufträge
  - Anzeigen Auftragspositionen
  - Hinzufügen von Bewegungsaufträgen

  - Ändern von Bewegungsaufträgen
  - Löschen von Bewegungsaufträgen
  - Hinzufügen von Auftragspositionen
  - Ändern von Auftragspositionen
  - Löschen von Auftragspositionen
  - CSV Dateien importieren
  - Löschen alles Bewegungsaufträge
- Kundenaufträge
  - Anzeigen von Kundenaufträge
  - Hinzufügen von Kundenaufträgen
  - Ändern von Kundenaufträgen
  - Bewegungspositionen in einem Auftrag anzeigen
  - Löschen von Kundenaufträgen
  - Löschen aller Kundenaufträge
- Bestand
  - Anzeigen des Bestands (nur gültige Bestandspositionen)
  - Anzeigen aller Bestandspositionen (z.B. auch von Positionen ohne zugewiesene LE)
  - Ändern von Bestandsposition
  - CSV Datei Importieren
  - Löschen von Bestandspositionen
  - Löschen aller Bestandspositionen
- Ladeeinheiten
  - Anzeigen von Ladeeinheiten
  - Ändern von Ladeeinheiten
  - CSV Datei importieren
  - Löschen von Ladeeinheiten
  - Löschen aller Ladeeinheiten
- Suchfunktion
  - Suche nach Aufträge
  - Suche nach Positionen
- Startseite zeigt eine Überischt über gespeicherte
  - Telegramme
  - Testfälle
  - Avisierte Positionen

  - Bestände
  - Bewegungsaufträge
- Fehlerbehandlung und Erzeugung sinnvoller Fehlermeldungen
- Prüfung aller übergebener Parameter auf Gültigkeit (z.B. keine Semikola, damit keine SQL Injections möglich sind, realisiert über einen regulären Ausdruck)
- Sende und Empfangsmodul
  - Senden eines Telegrammtextes an die Simulation
  - Empfangen eines Telegramms von der Simulation
  - Berücksichtigung von Quittungstelegrammen

Im Folgenden werden einige dieser Funktionalitäten exemplarisch vorgestellt und mit Bildern illustriert.

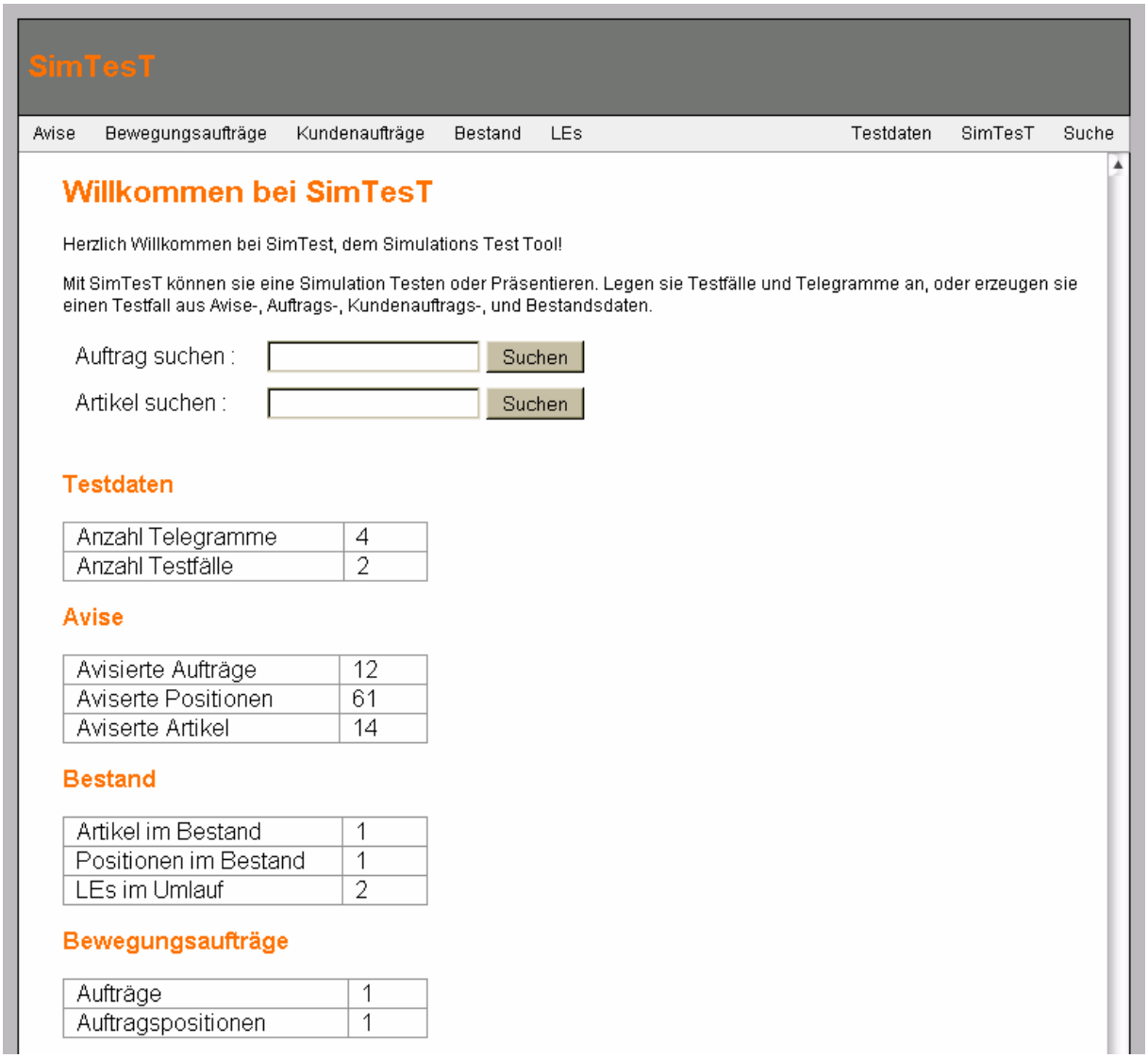


Abbildung 7.1 *Startseite von SimTesT (Screenshot)*

Die Startseite ist in Abbildung 7.1 gezeigt. Die Menüleiste ermöglicht den Zugriff unterschiedliche Funktionen (siehe Abbildung 7.2), diese wurden über CSS realisiert. Die Suchfelder ermöglichen eine Suche nach Aufträgen oder Artikeln. Eine Übersicht stellt im System enthaltenen Daten darunter dar.

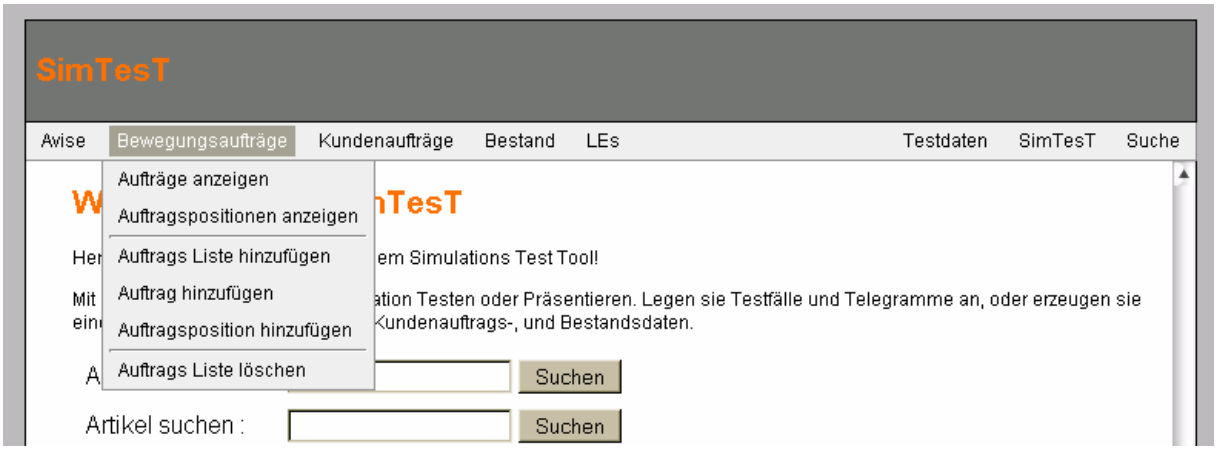


Abbildung 7.2 *Ausgeklapptes Menu „Bewegungsaufträge“ (Screenshot)*

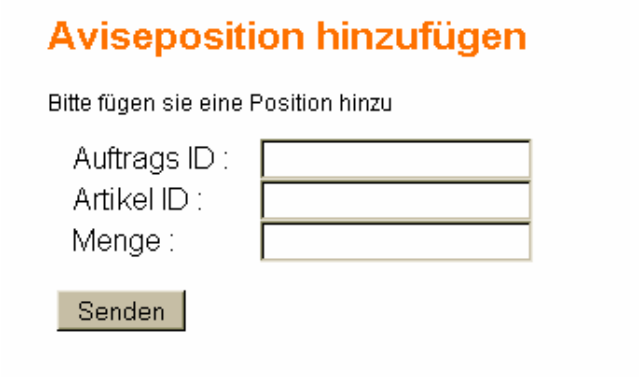


Abbildung 7.3 *Eine Aviseposition hinzufügen (Screenshot)*

Fügt man eine neue Position in die Aviseliste ein müssen Auftrags ID, Artikel ID und die erwartete Menge angegeben werden. Es ist möglich mehrere Positionen gleichzeitig anzulegen. Dazu gibt es die Möglichkeit CSV Dateien zu importieren. In einer CSV Datei werden drei Spalten mit den Angaben zu Auftrags ID, Artikel ID und erwartete Menge in der Datei erwartet (siehe Abbildung 7.4).

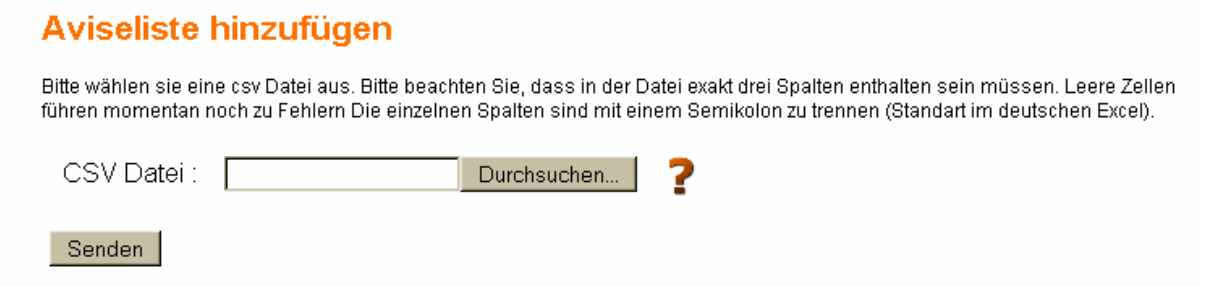
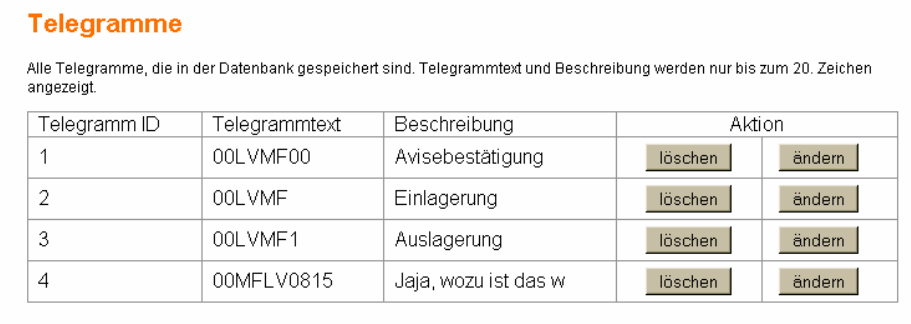


Abbildung 7.4 *Hinzufügen einer Liste avisierter Positionen in SimTesT (Screenshot)*

Eine Telegrammübersicht ist in Abbildung 7.5 gezeigt. Um die Übersichtlichkeit sicherzustellen werden von der Beschreibung und dem Telegrammtext nur die ersten 20 Zeichen angezeigt.



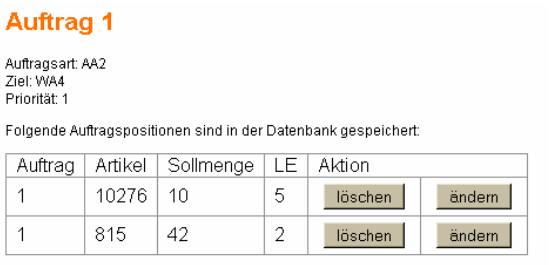
Telegramme

Alle Telegramme, die in der Datenbank gespeichert sind. Telegrammtext und Beschreibung werden nur bis zum 20. Zeichen angezeigt.

Telegramm ID	Telegrammtext	Beschreibung	Aktion	
1	00LVMF00	Avisebestätigung	löschen	ändern
2	00LVMF	Einlagerung	löschen	ändern
3	00LVMF1	Auslagerung	löschen	ändern
4	00MFLV0815	Jaja, wozu ist das w	löschen	ändern

Abbildung 7.5 *Telegrammübersicht in SimTesT (Screenshot)*

Wählt man „ändern“, kann man die Eigenschaften eines Telegramms ändern. Telegrammtext und Beschreibung sind dann in voller Länge sichtbar.



Auftrag 1

Auftragsart: AA2
Ziel: WA4
Priorität: 1

Folgende Auftragspositionen sind in der Datenbank gespeichert:

Auftrag	Artikel	Sollmenge	LE	Aktion	
1	10276	10	5	löschen	ändern
1	815	42	2	löschen	ändern

Abbildung 7.6 *Ein Bewegungsauftrag in SimTesT (Screenshot)*

Einen Bewegungsauftrag zeigt Abbildung 7.6. Zu sehen sind die Auftragsart (Auslagerauftrag), das Ziel, die Priorität, sowie die zum Auftrag gehörenden Telegramme. Für jedes Telegramm sind Auftragsnummer, Artikel ID, Sollmenge und die ID der Ladeeinheit (z.B. die Palette) angegeben. Es besteht die Möglichkeit eine Position zu löschen oder zu ändern.

7.2. Persönliches Fazit zur eingesetzten Software

Der Einsatz von Python als Programmiersprache hat sich als effizient erwiesen. Es lies sich recht schnell erlernen und die Implementierung des Testtool war zügig mit Python zu realisieren. Einen wesentlichen Anteil daran trägt die Templateengine Cheetah. Der Webserver Apache und die Datenbank MySQL haben sich (wieder einmal) als zuverlässig erwiesen.

Das Testtool webbasiert zu gestalten scheint im Nachhinein unglücklich, da im Testfall sowohl ein Webserver, als auch eine Datenbank, Python und die Templateengine Cheetah installiert und eingerichtet sein müssen. Dies stellt einen erheblichen Aufwand dar. Eine „stand alone“ Lösung wäre sicher einfacher einzurichten und würde. Die webbasierte Lösung bietet gegenüber einer Lösung mit einem konventionellen GUI keine nennenswerten Vorteile.

Verzeichnisse

Literaturverzeichnis

Kürzel	Verfasser	Quelle
ABE-08	Mark Aberdour	http://www.opensourcetesting.org/ 01.04.2008
AJE-08	AJE Consulting	http://www.aje.de/produkte.htm, 09.02.2008
ARN-02	D. Arnold, H. Isermann, A.Kuhn, H. Tempelmeier	Handbuch Logistik, VDI/Springer 2002
AUT-08	AutoMod	http://www.promodel.com/, 06.04.08
BAL-01	Prof. Dr-Ing. habil. Helmut Balzert	Lehrbuch der Software-Technik Band 1, 2. Auflage, Spektrum Akademischer Verlag, 2001
BAL-08	J. Ballhaus	http://www.warehouse-logistics.com/download/DE_Heft_SALT_2_2005_1.pdf, 06.04.08
BAL-98	Prof. Dr-Ing. habil. Helmut Balzert	Lehrbuch der Software-Technik Band 2, 1. Auflage, Spektrum Akademischer Verlag, 1998
BLG-02a	Dr. U. Ochs	Pflichtenheft, Materialflussstreuerung und Anlagenverwaltung, Geschäftsvorfälle Gesamtsystem, Böblingen 2002
BLG-02b	Dr. U. Ochs	Pflichtenheft, Materialflussstreuerung und Anlagenverwaltung, Schnittstelle LVS, Böblingen 2002
CHA-08	Channelpartner	http://www.channelpartner.de/technik/1709514/index4.html, 06.04.08
CHA-08a	Channelpartner	http://www.channelpartner.de/technik/1709514/index5.html, 06.04.08
DUA-08	DUAL	http://www.dualis-it.de/index.php?pageid=364, 06.04.08
DUS-01	Elfride Dustein, Jeff Rashka, John Paul	Software automatisch Testen, Springer 2001

EM-08P	Em Plant	http://emplant.de, 06.04.2008
ENT-08	Enterprise Dynamics	http://enterprisedynamics.com/, 06.04.2008
FLE-08	Flexsim	http://www.flexsim.com, 05.04.2008
FRA-08	Frauenhofer Institut	http://www.iml.fraunhofer.de/367.html, 09.03.2008
GLE-08	Prof. Dr. Harald Gleißner	Logistik, Grundlagen – Übungen – Fallbeispiele, Babler Verlag, GWV Fachverlage GmbH, Wiesbaden 2008
HOM-08	Prof. ten Hompel	Taschenbuch der Logistik, Springerverlag Berlin Heidelberg, 2008
INC-08a	inconso AG	http://c1.inconso.de/contell/cms/c1web/inconso/site/Softwareloesungen/Warehouse_Management/inconsoWMS_HELAS/index.html, 09.02.2008
INC-08b	inconso AG	http://c1.inconso.de/contell/cms/c1web/inconso/site/Softwareloesungen/Warehouse_Management/inconsoWMS_ILoS/index.html, 09.02.2008
INC-08c	Inconso AG	http://c1.inconso.de/contell/cms/c1web/inconso/site/Softwareloesungen/Warehouse_Management/StoreLiner/index.html, 09.02.2008
INT-08	Intersystems	http://www.intersystems.de/casestudies/cache/SD%20-ITDirector-TVA%20Langenscheidt.pdf, 12.04.2008
IPC-08	IPcomm	http://www.ipcomm.de/protocols_de.html, 02.04.2008
KAH-98	Prof. Dr. Bernd Kahlbrandt	Software-Engineering. Objektorientierunge Software-Entwicklung mit der Unified Modeling Language, 1. Auflage, Springer Verlag. Heidelberg 1998
KEY-03	Jessica Keyes	Software Engineering Handbook, Auerbach, 2003

KLA-04	Peter Klaus, Winfried Krieger	Lexikon Logistik, 3. Auflage, Gabler, Wiesbaden 2004
KOE-03	Reinhard Koether	Taschenbuch der Logistik, 2. Auflage, Fachbuchverlag Leipzig im Carl Hanser Verlag, Südtirol 2003
LAN-08	Lanner	http://www.lanner.com/, 06.04.2008
LIN-02	Ingo Linkweiler	Eignet sich die Skriptsprache Python für schnelle Entwicklungen im Softwareentwicklungsprozess ? http://www.ingo-linkweiler.de/diplom/Diplomarbeit.pdf, 11.03.2008
LOG-04	Logistik Heute	Logistik Heute 26/04 Nr. 1-1
LOG-08	Logistik Lexikon	http://www.logistik-lexikon.de/?main=/ccLiid160, 09.04.2008
LUD-07	Prof. J. Ludewig, Prof. H. Lichter	Software Engineering, Grundlagen, Menschen, Prozesse Techniken, 1. Auflage, dpunkt.Verlag, Heidelberg 2007
MAR-04	Heinrich Martin	Transport und Lagerlogistik, 5. Auflage, Vieweg, Wiesbaden 2004
MOE-07	Prof. Dr. Ralf Möller	Skript Software Engineering der TUHH (zu finden unter http://www.sts.tu-harburg.de)
NET-08	Netcraft	Netcraft: November 2006 Archiv, http://news.netcraft.com/archives/2006/11/index.html, 08.04.2008
NIS-08	Nissen & Velten	http://www.nissen-velten.com/front_content.php?client=3&lang=3&idcat=70&idart=129, 09.02.2008
NOC-08	Prof. Dr. Bernd Noche	Simulationsgestützte Testumgebungen, http://www.sim-serv.com/pdf/whitepapers/whitepaper_59.pdf,

		19.01.08
PAE-04	Prof.Dr. Barbara Paech Lars Borner, Jürgen Rückert	http://www-swe.informatik.uni-heidelberg.de/teaching/ss2004/ss2004SWE/swevorlss04Integrationstest.pdf, 12.03.2008
PAW-07	Prof. Dr. Günther Pawellek	Planung logistischer Systeme, Simulation in der Logistik, WS07/08
PAW-07a	Prof. Dr. Günther Pawellek	Logistik 1 – Materialflusssysteme, Lagerlogistik, WS07/08
PAW-04	Prof. Dr. Günther Pawellek, Dr. Axel Schönknecht	Logistik für Unternehmen 9/2004, Simulation in Planung und Betrieb logistischer Systeme
PFO-00	Hans-Christian Pfohl	Logistiksysteme, 7. Auflage, Springer, Heidelberg 2000
PRO-08	Pro Model	http://www.promodel.com/, 06.04.08
PSI-02	PSI logistics	Warehouse Management Software von PSI logistics, Berlin 2002
SAP-08	SAP AG	http://www.sap.com/germany/media/50082163.pdf, 09.02.2008
SAP-08a	SAP AG	http://searchsap.techtarget.com/sDefinition/0,,sid21_gci852485,00.html, 06.03.2008
SDZ-08	SDZ GmbH	http://www.sdz.de/16.0.html, 06.04.08
SIM-08	SimPlan AG	http://www.simul8.de/simul8/ubersicht.html, 06.03.08
SPA-08	SpaceNet	http://www.space.net/download/produktblatt/2/webdatenblatt_hosting_lasttest.pdf, 11.04.2008
SPI-06	Dr. Sven Spieckermann	http://ishk1.informatik.htw-dresden.de/lehre/ws06/Sim/sim_82_060127_8SimWorkshop_Dresden.pdf, 09.03.2008
SPI-04	Dr. Sven Spieckermann	Diskrete und Ereignisorientierte Simulation in Produktion und Logistik – Herausforderungen und Trends
SWI-08	Swisslog	http://www.swisslog.com/de/wds-index/wds-sw/wds-sw-wm.htm, 06.02.2008

SYS-08	Sysmat	http://www.sysmat.de/inhalt/produkte/matwms/, 04.04.2008
TES-08	testabillity.com	http://www.testability.com/Reference/Glossaries.aspx?Glossary=Testability, 08.04.2008
WAH-04	Renate Wahrig-Burfeind	Wahrig Fremdwörterlexikon 1. Auflage, Bertelsmann Lexikon Institut; Güthersloh, München 2004)
WIK-08	Die Wikipedia Gemeinschaft	Perl, http://de.wikipedia.org/wiki/Perl, siehe Obfuscation 06.04.2008
WOL-07	Oliver Wolf, Günther Dietze, Damian Daniluk	Software in der Logistik, Kapitel Warehouse Management Systeme
XJT-08	XJ Technology	http://www.xjtek.com/anylogic/, 06.04.08
ZEP-08	Zephyr	http://media.yourzephyr.com/images/screenshots/zephyr_ss_hires.zip, 12.04.2008
ZOL-06	Hand-Dieter Zollondz	Grundlagen Qualitätsmanagement. Einführung in Geschichte, Begriffe, Systeme und Konzepte, 2. Auflage, Oldenbourg, 2006

Abkürzungsverzeichnis

CORBA	Common Object Request Broker Architecture (Definiert durch die OMG)
EDI	Electronic Data Interchange
ERP	Enterprise Resource Planing
FT	Fördertechnik
IP	Internet Protocol
IPC	Inter Process Communication
LE	Lagereinheit
LTT	Ladungsträgertyp
LVS/WMS	Lagerverwaltungssystem/Warehouse Management System
MDT	Mobiles Daten Terminal
MHD	Mindesthaltbarkeitsdatum
OMG	Object Management Group
OSI	Open System Interconnection Reference Model
PPS	Produktionsplanung und-steuerung
QS, QA	Qualitätssicherung, Quality assurance
QM	Qualitätsmanagement
RGB	Regalbediengerät
RPC	Remote Procedure Call
SQL	Structures Query Language
TCP	Transmission Control Protocol
TE	Transporteinheit
WA	Warenausgang
WE	Wareneingang
WWS	Warenwirtschaftssystem

Tabellenverzeichnis

Tabelle 2.1 *Auflistung von Lagerverwaltungssysteme* 24
Tabelle 2.2 *Telegramme im Safeline Standard* 30
Tabelle 2.3 *Telegramme im Safeline Standardl (LVS -> MFR)* 30
Tabelle 2.4 *Telegramme 01, 04, 05 im Safeline Standard (LVS -> MFR)* 31
Tabelle 2.5 *Telegramme im Safeline Standrds (MFR -> LVS)* 32
Tabelle 2.6 *Beispiel für Äquivalenzklassen von Testfällen* 49
Tabelle 3.1 *Simulationstools (Auswahl)* 65
Tabelle 6.1 *Vor- und Nachteile der Skriptsprachen PHP, Perl und Python* 84
Tabelle 6.2 *In der Studienarbeit eingesetzte Software* 88

Abbildungsverzeichnis

Abbildung 1.1 *Logistikzentrum der Kühne + Nagel International AG (Quelle Wikimedia Commons)* 6
Abbildung 1.2 *Simulationstool Enterprise Dynamics (2D Modell, Screenshot)* 8
Abbildung 1.3 *Testsituation (eigene Darstellung)* 9
Abbildung 1.4 *Testreihenfolge (eigene Darstellung)* 10
Abbildung 2.1 *Beispiel für Lagerzonen (eigene Darstellung)* 14
Abbildung 2.2 *Lagerlayout eines Ersatzteillagers der Kion Group (ehemals Linde, Quelle: Arbeitsgebiet Technische Logisistik)* 15
Abbildung 2.3 *Bereiche in Lägern, nach [KLA-04 S.265] (eigene Darstellung)* 16
Abbildung 2.4 *Ebenen der Lagersteuerung (nach BAL-08 und GLE-08 S.217, eigene Darstellung)* 18
Abbildung 2.5 *Maske des incomsoWMS Standard (Bestandskorrektur)* 19
Abbildung 2.6 *Kommunikationspartner eines LVS (eigene Darstellung)* 21
Abbildung 2.7 *Buchungsarbeitsplätze im Wareneingang, Lufthansa Technik Hamburg (eigens Foto)* 22
Abbildung 2.8 *Fördertechnik bei der Lufthansa Technik Hamburg (eigenes Foto)*..... 27
Abbildung 2.9 *Lagervisualisierung im inconso StandardWMS (Quelle: Programmbeschreibung)* 27
Abbildung 2.10 *Anzahl der Fehler vor und nach einer Prüfung/Fehlerbehebung (eigene Darstellung)* 33
Abbildung 2.11 *Qualität nach ISO 55 350 (eigene Darstellung) (eigene Darstellung)*..... 36
Abbildung 2.12 *Qualitätskreis (nach [ZOL-06, S. 180], eigene Darstellung)* 37
Abbildung 2.13 *Zusammenhang zwischen Behebungskosten und Entdeckungszeitpunkt eines Fehlers ([KAH-98 Seite 43], eigene Darstellung)* 39
Abbildung 2.14 *Zeit-Kosten-Qualität (nach KAH-98 S. 42)* 41
Abbildung 2.15 *Unterschied zwischen Blackbox- und Whiteboxtest (eigene Darstellung)* 43
Abbildung 2.16 *Grundsätzlicher Testablauf (nach [LUD-07] eigene Darstellung)* 50
Abbildung 2.17 *Testgeschirr (nach [PAE-04] S.5, eigene Darstellung)*..... 51
Abbildung 2.18 *Zephyr, ein Test Management Tool [ZEP-08]* 54
Abbildung 2.19 *SpaceNet_Lasttest Lasttestool [SPA-08]*..... 55
Abbildung 3.1 *Klärungsband (NiO) bei Lufthansa Technik Hamburg (eigenes Foto)*..... 64
Abbildung 3.2 *Schilder über dem Klärungsband, NiO (eigenes Foto)*..... 64
Abbildung 3.3 *Das Simulationstool Flexsim (3D Modell, Screenshot)*..... 66

Abbildung 3.4 *Testsituation: Die Simulation kommuniziert mit dem Testtool statt mit dem LVS (eigene Darstellung)*..... 67
Abbildung 3.5 *Das TCP Test Tool (Screenshot)*..... 70
Abbildung 4.1 *Use Case Testsystem: Telegramme und Testfälle (eigene Darstellung)*.. 73
Abbildung 4.2 *Use cases für Avisedaten (eigene Darstellung)*..... 75
Abbildung 5.1 *Die Klasse Telegramm (eigene Darstellung)*..... 78
Abbildung 5.2 *Die Klasse Testfall (eigene Darstellung)*..... 78
Abbildung 5.3 *Aktivitätsdiagramm: Daten in der Datenbank speichern (eigene Darstellung)* 81
Abbildung 7.1 *Startseite von SimTesT (Screenshot)*..... 92
Abbildung 7.2 *Ausgeklapptes Menu „Bewegungsaufträge“ (Screenshot)*..... 93
Abbildung 7.3 *Eine Aviseposition hinzufügen (Screenshot)*..... 93
Abbildung 7.4 *Hinzufügen einer Liste avisierter Positionen in SimTesT (Screenshot)*.... 93
Abbildung 7.5 *Telegrammübersicht in SimTesT (Screenshot)*..... 94
Abbildung 7.6 *Ein Bewegungsauftrag in SimTesT (Screenshot)*..... 94

Code Verzeichnis

Code 6.1 *Beispiel für „write only“ Code in Perl [WIK-08]* 85
Code 6.2 *Beispiel für Codevarianten in Perl* 86
Code 6.3 *Beispiel für ein Cheetah Template* 87

Code Formatierung

Die Formatierung des Codes orientiert sich an dem Dokument „Code Konvention Issue 1-1“. Vielen Dank an Torben Becker, der dieses freundlicherweise zur Verfügung gestellt hat. Aufgrund der Eigenschaften von Python mussten wenige Änderungen vorgenommen werden. Daher sind die wichtigsten Konventionen hier noch einmal aufgeführt. Es wird zwischen zwei Benennungsarten mit je zwei Varianten unterschieden. In der ersten Art werden alle Wörter eines Bezeichners zusammengeschrieben. In der zweiten Benennungsart werden einzelne Worte durch Unterstriche voneinander getrennt. Die beiden Varianten sehen vor, dass der erste Buchstabe groß bzw. klein geschrieben wird.

Name	**Beispiele**
Groß und zusammen	NameDerKlasse, VerbindungZurDatenbank
Klein und zusammen	nameDerFunktion, berechneFakultät
Groß und mit Unterstrichen	Name_Des_Bezeichners
Klein und mit Unterstrichen	name_Des_Bezeichners

Benennung von Klassen

Klassen werden „Groß und Zusammen“ geschrieben.

Benennung von Methoden

Methoden werden „Klein und Zusammen“ geschrieben

Benennung von Variablen

Lokale Variablen werden „Klein und Zusammen“ geschrieben. Globale Variable werden „Klein und mit Unterstrich“ geschrieben, damit man sie als solche erkennt. Diese Konvention steht im Gegensatz zu „Code Konventionen“, da Python Variablen in Modulen mit einem einzelnen Unterstrich nicht importiert.

Strings

Strings können in Python in einfache oder doppelte Anführungszeichen gesetzt werden. Für normale Strings werden einfache verwendet. Handelt es sich bei einem String um eine SQL Anweisung wird diese in doppelte Anführungszeichen gesetzt. Strings über mehrere Zeilen müssen in drei Anführungszeichen gesetzt werden.

SQL Anweisungen

Alle SQL Schlüsselwörter (SELECT, FROM, …) werden groß geschrieben, alle anderen Wörter klein.

Dateiendungen

Python Dateien enden auf .py, Templates auf .tmpl und die Konfigurationsdateien (von denen es zur Zeit nur eine gibt) auf .conf. Um eine leichte Wartbarkeit sicherzustellen werden keine HTML Dateien in diesem Projekt benutzt. Stattdessen werden Templates eingesetzt, die nach Bedarf zusammengesetzt werden, so ist es möglich Änderungen (z.B. im Menu) in nur einem Template zu machen. Die Formatierung ist in CSS festgehalten. Alle CSS Dateien enden auf .css.

Danksagung

Ich möchte mich bei den betreuenden Professoren Prof. Dr. Ralf Möller und Prof. Dr. Günther Pawellek, sowie ihren wissenschaftlichen Mitarbeitern Dr. Axel Schönknecht und Sebastian Wandelt für die fachliche Unterstützung bedanken. Ein großes Dankeschön geht auch an meine Familie, die mich während meines Studiums immer unterstützt hat und ohne die ich nicht dort wäre, wo ich heute bin. Außerdem möchte ich mich bei meiner Freundin Maggy Decka dafür bedanken, dass sie in den letzten Wochen so liebevoll an meiner Seite stand.