ALEA

Testautomatisierung

für Software mit einem SWT-Rich-Client

Staatliche Studienakademie Thüringen

Berufsakademie Gera

Kurs:	**PI07**
Von:	**Förster, Patrice**
Vorgelegt am:	**02.10.2008**
Studienbereich:	**Technik**
Studienrichtung:	**Praktische Informatik**
Matrikelnummer:	**G070135 PI**
Ausbildungsstätte:	**ALEA GmbH**
Betreuer:	**Dipl.-Ing. Thomas Roth**

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AWT – Abstract Window Toolkit

Ist eine Komponente zum Programmieren von Grafischen Oberflächen.

CRM – Customer Relationship Management

Es werden alle vertriebsrelevanten Geschäftsprozesse, Kommunikationskanäle und Unternehmensbereiche wie Akquisition, Marketing, Vertrieb und Service zusammengefasst.

DB2 – Database 2

Es handelt sich bei DB2 um eine von IBM entwickelte kommerzielle und relationale Datenbank, besonders für Großrechner aber auch für PCs. Dabei wird eine Vielzahl von Betriebs- und Großrechnersystem unterstützt.

EJB – Enterprise Java Beans

EJB ist eine serverseitige Komponentenarchitektur für die Java Plattform Enterprise Edition. Diese Technologie ermöglicht schnelle und vereinfachte Entwicklung von verteilten, transaktionalen, sicheren und portablen Anwendungen, die auf Java basieren.

ERP – Enterprise Resource Planning

Hierbei handelt es sich um eine betriebswirtschaftliche Software, die die Steuerung von betrieblichen Geschäftsprozessen ermöglicht. Es werden die Bereiche der Organisation, Verwaltung als auch der Kontrolle des Unternehmens abgedeckt.

GUI – Graphical User Interface

Hiermit ist die grafische Benutzeroberfläche eines Programms gemeint, über die eine Person das Programm steuert.

ID – Identifier

Ein Identifier ist ein Attributtyp, der ein deklariertes Attribut als einzigartiges Erkennungsmerkmal für die Elementinstanz, in der es verwendet wird, einsetzbar macht. Der Attributwert eines ID-Attributs muss eindeutig und einzigartig im Dokument sein. Es darf also niemals vorkommen, dass zwei Attribute dieselbe ID verwenden.

J2EE – Java 2 Enterprise Edition

Diese Java-Edition baut auf der Standard Edition auf und enthält zudem Industriestandards für die Implementierung einer unternehmensklassifizierte, dienstleistungsorientierte Architektur (engl. SOA) sowie von zukünftigen Webanwendungen.

JNI – Java Native Interface

Das ist eine standardisierte Schnittstelle um native Java Methoden und die virtuelle Maschine von Java in native Anwendungen des Betriebssystems einbetten zu können. Damit ist es ebenfalls möglich Betriebssystemaufrufe aus dem entwickelten Programm auszuführen.

RMI – Remote Method Invocation

Dies ist ein Mechanismus zur Kommunikation zwischen verteilten Java-Objekten. Es wird zu Erstellung von Java-Anwendungen verwendet, welche über ein Netzwerk miteinander kommunizieren können.

UI – User Interface

Das ist die Schnittstelle, worüber der Benutzer mit der verwendeten Software in Form von verschiedenen Aktionen wie Tastatureingaben oder Mausklicks kommunizieren kann.

URL – Uniform Resource Locator

Eine URL ist eine Zeichenkette, die die Position eines Programmes oder einer Datei im Internet definiert

1 Einführung

Das Ziel dieser Projektarbeit ist die Auswahl und Einführung eines UI-Testtools bei der ALEA GmbH für die Warenwirtschaftssoftware ALEA Commerce Suite zur Automatisierung von Funktionstests und zur Lasterzeugung bei Performancetests. Anhand dieser Tests kann anschließend nicht nur überprüft werden, ob alle Funktionalitäten einwandfrei fehlerfrei sind, sondern auch, ob die bereits vorhandene Hardware bei dem Kunden für ALEA Commerce Suite geeignet ist oder aufgerüstet werden muss. Die vorliegende Projektarbeit ist thematisch in zwei Teile gegliedert.

Es soll zunächst ein geeignetes Werkzeug zur Durchführung von User-Interface-Test (UI-Tests) ausgewählt werden. Dazu sollen zunächst bevorzugt OpenSource-UI-Testtools für Java-SWT-Clients (z.B.: GUIdancer oder Fit) evaluiert und mit kommerzieller Software, wie beispielsweise Mercury, Hewlett Peckard (HP) Loadrunner oder Seque Silkperformer verglichen werden. Sollte kein Open-Source-Werkzeug die Anforderungen der ALEA GmbH erfüllen, ist ein geeignetes kommerzielles Werkzeug auszuwählen. Folgende Kriterien müssen durch das UI-Testwerkzeug erfüllt sein: Es muss SWT- bzw. Eclipse-RCP-Clients unterstützen, Makros und Skripte leicht erstellen lassen, Auswertungswerkzeuge für die Tests zur Verfügung stellen und Multi-User-Support für Lasttests ermöglichen.

Nachdem ein geeignetes UI-Testtool für diese Aufgabe gewählt wurde, soll es bei ALEA GmbH eingeführt werden. Dazu sind entsprechende Tests für Hardware-Sizings und Standard-Funktionstests für ALEA Commerce Suite zu erstellen, auszuwerten und zu dokumentieren. Die erstellten Skripte sollen dann bei ALEA GmbH für ‚generelle Funktionstests' sowie für Lasttests für das Hardware-Sizing in Kundenprojekten genutzt werden.

2 Vorbetrachtungen zum Thema

In diesem Kapitel soll auf einige wichtige Dinge eingegangen werden, wie u. a. das Klären von grundlegenden Begriffen und Herangehensweisen, sodass verständlich wird, wie das Thema zu behandeln ist beziehungsweise welche Vorgänge und Abläufe für den erfolgreichen Abschluss der Thematik notwendig sind.

2.1 Arten des Testens

Es gibt außer den hier betrachteten Arten des Testens natürlich noch viele verschiedene andere wie beispielsweise den Regressionstest, jedoch werden hier nur diejenigen erläutert, welche auch dem Thema entsprechend sinnvoll sind.

2.1.1 Funktionstest

Der Funktionstest ist der Test einer in der Applikation implementierten Funktion. Es werden also Komponenten bzw. Subsysteme getestet. Funktionsbezogenes Testen ist eine gute Alternative zum ablaufbezogenen Testen. Beim Funktionstest wird jedoch nicht die innere Struktur getestet, sondern lediglich das Verhalten des Systems bei Ausführung der Funktion. Deswegen ist der Funktionstest ein Black-Box-Test, siehe Abbildung 1.

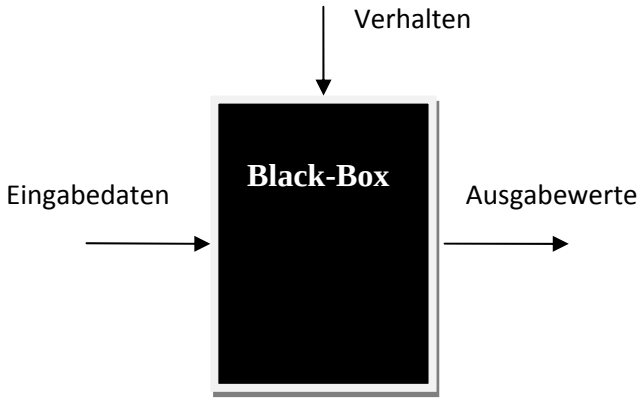


Abbildung 1: Der Black-Box-Test
Quelle: „vom Autor erstellt“

Im Gegensatz zum Black-Box-Test erfolgt die Auswahl der Tests immer auf Grundlage der vorangegangenen Implementierung. Das heißt, es wird die Struktur des Objektes und damit direkt der Quellcode getestet.

Der Entwurf der Tests basiert auf der Grundlage von vorher festgelegten Spezifikationen. Diese werden von einem Experten vor der Programmierung der Komponenten festgelegt. Aus diesen Spezifikationen können dann ebenso Testdaten erstellt werden, welche für die Ausführung der Tests notwendig sind. Hierbei ist nun wichtig, welche die zu erwartenden Ergebnisse (Nachbedingungen) sind, denn aufgrund der Spezifikationen gibt es folglich viele Eingabedaten, welche die Argumente (Vorbedingungen) darstellen. Dies ist in Abbildung 2 illustriert. Sie sind jeweils entsprechend so zu wählen, dass letztendlich jede mögliche Option dieser Funktion getestet werden kann. Denn je nach eingegebenen Daten finden andere Ereignisse statt. Die Funktion verhält sich somit anders, was wiederum die Ausgabedaten beeinflusst. Über die Verarbeitungsregel wird bestimmt, in welcher Art und Weise die Eingabedaten durch die Funktion verarbeitet werden. Es ist also ebenfalls auf eine korrekte Spezifikation der Datenflüsse zu achten. Eine beidseitige Betrachtung der Schnittstelle, sowohl vom Sender als auch vom Empfänger, ist vonnöten. Anhand von Ein- und Ausgabedaten kann verglichen werden, ob die Funktion den Anforderungen gemäß implementiert wurde und die Schnittstelle ordnungsgemäß funktioniert.

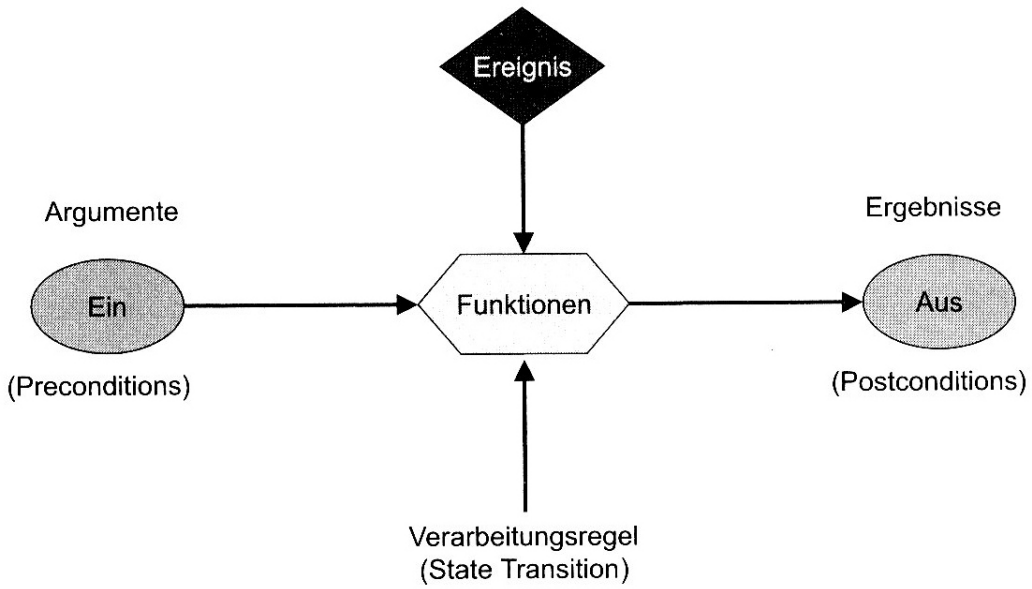


Abbildung 2: Funktionsbezogener Test

Quelle: [SW02], S.14

2.1.2 Lasttest

Der Lasttest ist die Belastung eines Anwendungssystems nach bestimmten Kriterien bis an ihre Grenzen, welche aber in der Praxis nie erreicht wird. Dadurch kann zum Beispiel die Infrastruktur oder das Systemverhalten zu bestimmten Zeitpunkten ermittelt werden. Mit Anwendungssystem ist dabei die Gesamtheit aus Clients, Servern, dem Netzwerk, den vorhandenen Datenbanken, sowie mögliche andere Komponenten, die für den Betrieb der zu testenden Applikation wichtig sind, gemeint. Das Anwendungssystem wird betrachtet, um definierte Aussagen über Antwortzeiten, den Datendurchsatz als auch die Stabilität der Anwendung treffen zu können. Es gibt bei solchen Tests mehrere Varianten, zum Beispiel ein Test mit der maximal möglichen Anzahl von gleichzeitig aktiven (konkurrierenden) Benutzern, ein Test zur allgemeinen Bestimmung des Systemverhaltens bzw. dessen spezielles Verhalten zu einem bestimmten Spitzenzeitpunkt.

Die Ziele eines Lasttests sind dabei wie folgt definiert:

- Optimale Ausnutzung der Anwendung in den Bereichen Leistung, Funktionsfähigkeit und Verfügbarkeit
- Identifizierung, zu welchem Zeitpunkt bestimmte Komponenten völlig aus- bzw. überlastet sind, Engpässe und Einbrüche in der Performance entstehen und ggf. Fehler in Funktionalitäten durch solche Belastungen auftreten
- Grenzen der Belastbarkeit für die Anwendung zu finden

Für die Vorbereitung und Durchführung eines Lasttests ist eine bestimmte Reihenfolge zwingend notwendig, welche wie folgt aufgeführt ist:

- *Festlegen der Anforderungen an die auszuführenden Geschäftsvorfälle*
- *Entwerfen der Testskripte*
- *Aufbau eines lasterzeugenden Systems*
- *Durchführung eines Lasttest*
- *Monitoring*[1] *der Systemkomponenten*
- *Analyse der Ergebnisse und Performanceoptimierung*
- *Reporting der Ergebnisse*

Quelle: [Wul08], S.16

[1] Kontinuierliche Analyse über einen gewissen Zeitraum

Da ein Lasttest eine wichtige und optimale Ergänzung zu funktionalen Tests darstellt, können die für den Funktionstest aus den spezifizierten Anforderungen entworfenen Tests für die Testskripte genutzt werden. Diese werden dann später automatisiert auf verschiedenen Rechnern ablaufen. So können dann Engpässe und Einbrüche in der Leistung aber auch Fehler, die nur unter Last entstehen, gefunden werden. In Abbildung 3 ist solch eine typische Vorbereitung für die Tests dargestellt.

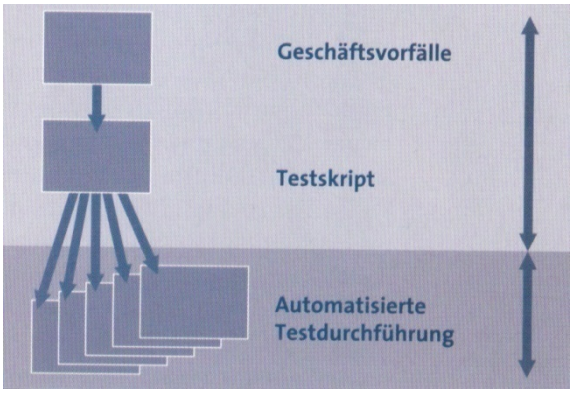


Abbildung 3: Testvorbereitung von Last- und Performancetests

Quelle: [Wul08], S.17

Nach Abschluss der notwendigen Vorbereitungen beginnt die eigentliche Durchführung, wobei der Aufbau eines lasterzeugenden Systems hier der erste Schritt ist. Solche ein Aufbau ist schematisch in Abbildung 4 dargestellt. Anfangs wird ein Last-und Performance-Controller benötigt, von dem aus der gesamte Prozess gesteuert wird. Über diesen Controller werden sogenannte lasterzeugende Agenten gestartet oder auch gestoppt, welche auf den Client-Rechnern zu finden sind. Die Aufgabe der Agenten ist es ein oder mehrere Instanzen der zu testenden Applikation auf einem Client-PC zu starten. Vom Controller aus können alle Agenten verwaltet werden. Da bereits die Testskripte geschrieben wurden, werden diese nun automatisiert auf den einzelnen Clients gestartet und somit die Last auf den Servern und Datenbanken erzeugt.

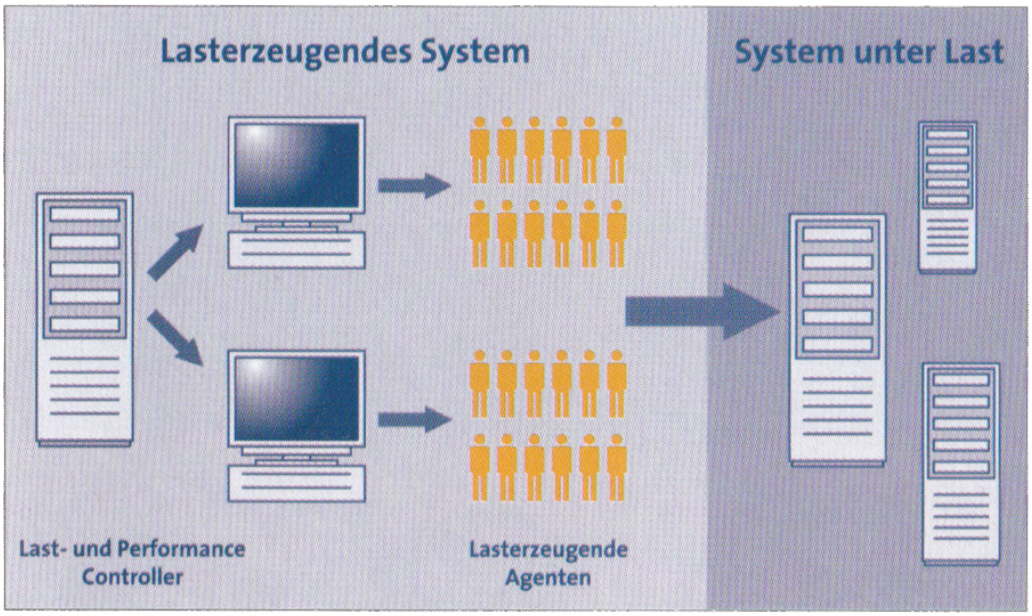


Abbildung 4: Aufbau eines lasterzeugenden Systems

Quelle: [Wul08], S.16

2.2 Testautomatisierung

Die Testautomatisierung ist die Durchführung von erstellten Tests, ohne dass dazu menschliche Tester benötigt werden, d.h. die Tests werden vollständig vom Computer ohne fremde Hilfe ausgeführt. Dabei ist es nicht zwingend möglich, dass jeder Test automatisierbar ist. Die Absicht der Testautomatisierung liegt darin, dass eine effiziente, langfristige Durchführung von Tests ermöglicht wird. Es ist dann möglich, den Personen, die vorher diese Aufgaben durch manuelle Tests übernommen hatten, nun andere, auch kreativere Aufgaben zuzuweisen. Der Prozess des Testens ist sehr aufwendig, langwierig und auch aufgrund des beschäftigten Personals teuer. Außerdem ist der Mensch als Tester sehr fehlerbehaftet, denn wenn er über einen längeren Zeitraum testet, kann dies unter Umständen zur Verringerung der Leistungsfähigkeit, zu Ermüdungserscheinungen und größerer Fehleranfälligkeit führen. Computer besitzen diese „Schwächen“ jedoch nicht.

Um aber eine komplette Automatisierung der Tests möglich zu machen, sind dafür große Investitionen für die nötigen Testverfahren sowie auch für die Lizenzierung und Nutzung des verwendeten Automatisierungs-Tools nötig. Dem gegenübergestellt hat solch eine Automatisierung auch einige Vorteile, welche nachfolgend beschrieben werden:

- Die Anwendung, welche getestet werden soll, kann in modifizierter Form trotzdem weiterhin getestet werden. Die Testprogramme können immer wiederverwendet werden. Sie müssen dann lediglich den Modifizierungen der Applikation angepasst werden.
- Der Ablauf der Tests erfolgt ohne menschliche Tester, d.h. der gesamte Vorgang wird vom Computer gesteuert.
- Dies wiederrum lässt die Ausführung von mehr Tests zu, als menschliche Tester überhaupt bewältigen könnten.
- Eine Automatisierung macht es außerdem möglich, dass viele Nutzer simuliert werden können, was sonst wegen Mangel an Testern gar nicht möglich wäre. Dies ist besonders für den Lasttest relevant.
- Weiterhin kann so auch die Applikation auf verschiedenen Hard- sowie Softwarekonfigurationen (hiermit sind Betriebssysteme wie Windows, Linux, Mac OS gemeint) getestet werden.
- Nicht zu vergessen ist die Verringerung des zeitlichen Aufwandes. Bis zur vollständigen Automatisierung der Tests wird zwar auch viel Zeit benötigt, jedoch wird bei erfolgreicher Implementierung des gewählten Werkzeugs und der Automatisierung der Tests eine große Zeitersparnis erreicht. Somit kann die Anwendung früher veröffentlicht und vor der Konkurrenz auf den Markt gebracht werden.

Zum erfolgreichen Testen einer Software sind dabei drei wesentliche Schritte zu beachten:

1. Der Entwurf der zu automatisierenden Vorgänge
2. Die Ausführung der automatisierten Tests
3. Vergleich der vorhandenen Ergebnisse

Im ersten Schritt erfolgen zunächst die Spezifikation der Anforderungen sowie das Bereitstellen der Testskripte. Weiterhin werden hier ebenso die erwarteten Ergebnisse festgelegt. Schritt zwei beinhaltet die eigentliche Automatisierung. Hier wird die programmierte Anwendung ausgiebig getestet. Dies erfolgt in einer bestimmten Zeitspanne. Je nach Größe des Softwareprojekts gibt es in der Applikation mehr zu testen. Deshalb ist die Dauer der Tests davon abhängig und sie kann nur Minuten, aber auch Wochen oder sogar Monate betragen. Nach erfolgtem Test können nun die durch den Test erhaltenen mit den erwarteten Ergebnissen verglichen werden. Der dritte Schritt ist nicht zwingend automatisiert,

aber es sollte nach Möglichkeit eine Funktion für solche Vergleiche im Tool vorhanden sein, welches für die Testautomatisierung gewählt wurde. Aus diesen Vergleichen und den daraus resultierenden Protokollen ist nun ersichtlich, ob unbekannte Fehler entdeckt wurden.

In der ALEA GmbH ist es ebenso nicht möglich, dass die ALEA Commerce Suite komplett automatisiert getestet werden kann. Der Grund liegt in den zu hohen Anpassungskosten in der Software und den sehr komplex aufgebauten Strukturen der einzelnen Komponenten. Deswegen werden nur zentrale Funktionen, welche im Sizing bzw. für die Standard-Funktionstest überprüft werden sollten, automatisiert.

2.3 Sizing

Die präzise Ermittlung der Hardwareanforderungen für ein Softwareprojekt wird als Sizing bezeichnet. Auch können dadurch bestimmte Performanceprobleme erkannt werden. Als Grundlage hierfür wird ein Lasttest, siehe Abschnitt 2.1.2, verwendet. Ein Sizing erfolgt immer mit einer fest definierten Anzahl an Benutzern, beispielsweise 10, 100 oder auch 1000 Benutzer. Diese Benutzerzahl ist möglichst an der Realität orientiert, damit ebenso brauchbare Ergebnisse erzeugt werden können. Mit den Benutzern werden nun Geschäftsvorfälle getestet, die für den jeweiligen Kunden typisch sind und auch in der Realität sehr häufig benötigt werden. Anhand der Ergebnisse des Sizings kann ermittelt werden, ob die Hardware für die Applikation sowie andere zusätzliche Software, zum Beispiel eine Datenbank, welche für die Anwendung benötigt wird, ausreicht. Ist dies nicht der Fall, wird eine neue Hardwarekonfiguration zusammengestellt beziehungsweise die bestehende erweitert und nochmals getestet. Dieser Ablauf wird solang durchlaufen, bis die Kriterien für einen ordentlichen Betrieb der Software erfüllt sind. Mit Hilfe des Sizings kann für jeden Kunden der Software spezifisch die Hardware bestimmt werden, da jeder von ihnen die Anwendung in anderer Weise benutzt, zum Beispiel wird eine Funktion wie Auftragserfassung oder Stornierung viel öfter verwendet.

2.4 Die Rich-Client-Platform

Um eine Rich-Client-Anwendung erstellen zu können, ist eine bestimmte Reihe von Plugins notwendig. Dies zusammengefasst wird auch als Rich-Client-Platform (RCP) bezeichnet. Da

Eclipse auf der Rich-Client-Platform basiert, war das Eclipse Integrated Development Environment (IDE) somit die erste Rich-Client-Applikation. Folgend werden die Funktionalitäten, die die Rich-Client-Platform charakterisieren, sowie die Architektur der Eclipse-RCP, welche in Abbildung 5 dargestellt ist, beschrieben:
Die Plattform besitzt eine dynamische Plugin-Architektur und lässt sich beliebig um Funktionen und Designs erweitern. Einen wesentlichen Anteil der Rich-Client-Platform machen die GUI-Komponenten aus. Die in der Abbildung 5 zu sehenden Komponenten Standard Widget Toolkit (SWT) sowie JFace enthalten alle Grundfunktionalitäten für die Erstellung von grafischen Oberflächen. Das UI wird durch sogenannte Toolkits und Extension Points erstellt. Mit Hilfe vom SWT ist es möglich, Rich-Client-Benutzeroberflächen zu erstellen, die genauso aussehen und sich auch entsprechend verhalten wie die des verwendeten Betriebssystems. Dies wird dadurch erreicht, dass die SWT-Elemente auf den nativen Elementen des Graphical User Interfaces (GUI) des Betriebssystems aufsetzen, siehe hierzu Abschnitt 2.5. Durch diese Bibliotheken baut sich die generische Workbench auf. Komponenten dieser Workbench sind zum Beispiel Editoren oder Views (Ansichten). Als Bestandteil der Rich-Client-Platform sind sie folglich in jeder RCP-Anwendung verfügbar. Ebenfalls in Eclipse 3.0 wurde die OSGi-Plattform eingeführt. Dies ist eine standardisierte Umgebung zum Ablauf von Java Modulen, die Bundles genannt werden. Mit Hilfe von OSGi ist es sogar möglich Bundles direkt im Betrieb austauschen zu können. Wie der Name Rich-Client-Platform bereits aussagt, handelt es sich um Client-Applikationen. Jedoch ist es mit RCP möglich, dass sowohl online als auch offline gearbeitet werden kann, falls zum Beispiel einmal die Verbindung zum Server nicht hergestellt werden kann. Das Offline-Arbeiten wird durch eine lokale Datenhaltung (persistence layer) und eine Anwendungslogik bei dem Client sichergestellt. Weiterhin gibt es Komponenten für Konnektivität als auch für die Client-Server-Synchronisierung. So können die Daten vom Client mit dem Server abgeglichen werden, sobald eine Verbindung erfolgreich aufgebaut wurde. Die Synchronisierung verhindert beispielsweise verschiedene vorhandene Dateiversionen. Es ist außerdem durch entsprechende Plugins sogar möglich, Grafik- und Office-Anwendungen in die eigene RCP-Applikation zu integrieren.

Damit der Benutzer nicht die Installation und Konfiguration selbst bewältigen muss, sollte dies automatisiert vonstatten gehen. Dank der Plugin-Architektur als auch der Update-Komponente von Eclipse gibt es Möglichkeiten zur vollständigen Automatisierung von Installation, Konfiguration und Aktualisierung der Client-Anwendung. Um RCP-

Anwendungen erstellen zu können, gibt es eine Vielzahl von Tools. Hier seien lediglich der Wizard für Plugin-Projekte sowie der Editor für die Produktkonfiguration als die beiden wichtigsten genannt.

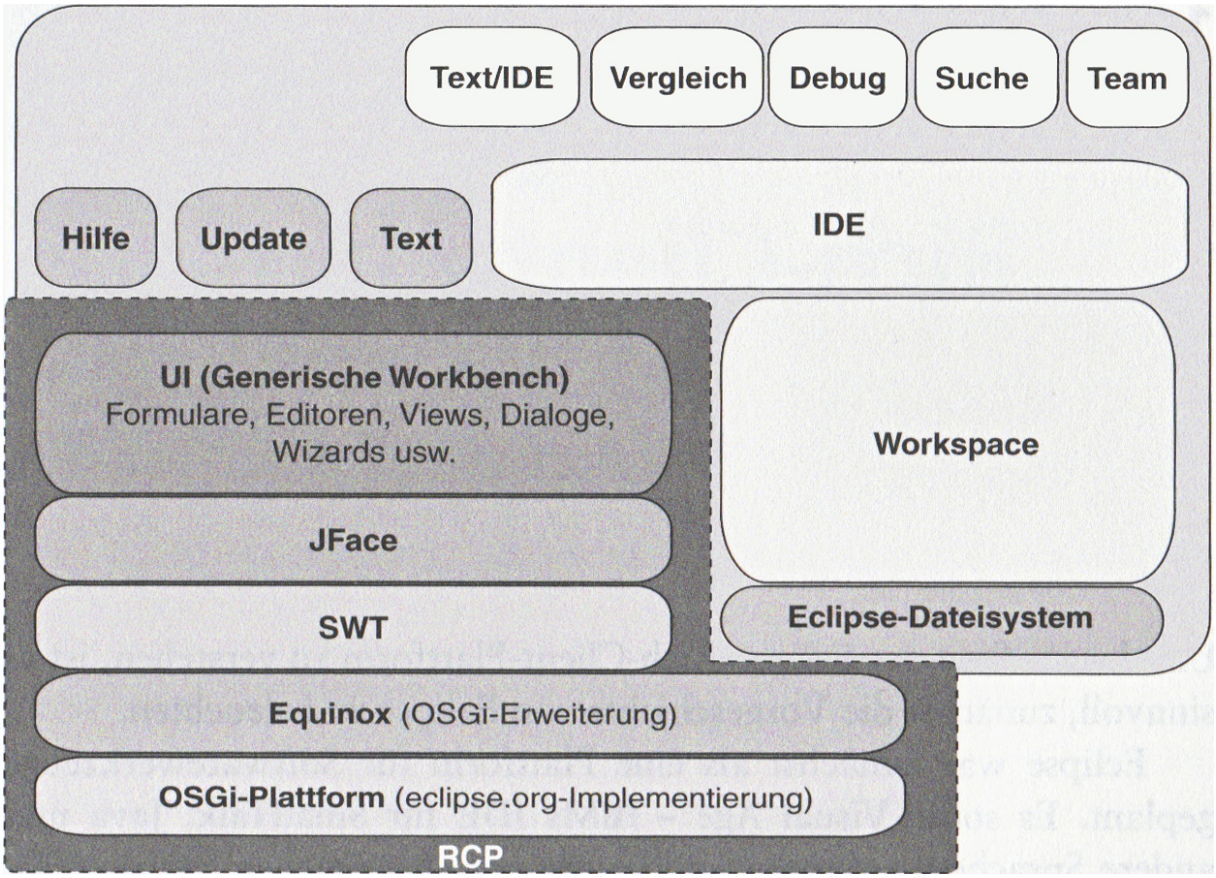


Abbildung 5: Die Eclipse-Plattform

Quelle: [Dau07]

2.5 SWT – The Standard Widget Toolkit

Das SWT ist eine für jeden frei verfügbare (Open Source) Softwarekomponente, die native Widget-Funktionalitäten für RCP und somit auch für Eclipse zur Verfügung stellt. Sie ist dabei komplett in Java implementiert worden. Das Abstract Window Toolkit (AWT) und Swing sind analog zu SWT, jedoch emulieren sie die GUI-Komponenten in Java. Bei SWT hingegen werden native Widgets verwendet. Das heißt im Einzelnen, dass die SWT-Komponenten direkt auf die GUI-Elemente des Betriebssystems aufsetzen. Daher sehen Applikationen, die entsprechend mit SWT implementiert wurden, aus wie jene des verwendeten Betriebssystems, beispielsweise Windows, Linux oder Mac OS X. Dies spiegelt sich auch im Ressourcenverbrauch wieder, denn SWT verbraucht deutlich weniger als zum

Beispiel Swing. Die Komponenten der SWT-Klassen sind sehr robust und haben eine sehr hohe Ansprechgeschwindigkeit, was sich vor allem bei großen Softwareprojekten bemerkbar macht. Mit Hilfe von Java Native Interfaces (JNI) stellt SWT sogar eine Schnittstelle zur Programmiersprache C bereit, welche verwendet werden kann, um direkte Aufrufe zum Betriebssystem zu ermöglichen.

Das SWT besteht aus sogenannten Widgets. Dies sind kleine Bausteine, um eine Benutzeroberfläche implementieren zu können. Wie in Abbildung 6 zu sehen, ist eine relativ große Anzahl solcher Widgets vorhanden, wobei aber im SWT-Packet auch nichtnative Komponenten enthalten sind. Diese sind entsprechend am Anfang des Namens mit einem C gekennzeichnet, was mit dem Package org.eclipse.swt.custom zusammenhängt. Die oberste aller SWT-GUI-Elemente ist die Klasse Widget; diese ist abstrakt.

Die Einteilung der einzelnen Funktionen, welche SWT bereitstellt, ist über Packages geregelt, welche überall in Java ihre Anwendung finden. Die Benennung ist laut Namenskonvention org.eclipse.swt. So existiert zum Beispiel ein Package org.eclipse.swt.layout, welche alle Klassen für die Implementierung von verschiedenen Layouts enthält. Natürlich gibt es auch weitere wie beispielsweise das für die Ereignisverarbeitung, was den zentralen Mechanismus zur Kommunikation zwischen der erstellten Anwendung und der grafischen Oberfläche darstellt. Hierbei wird von der Anwendung ein Listener mit einem Widget registriert. Dieser Listener reagiert bei einer ausgeführten Aktion, unter anderem Mausklicks oder Eingaben auf der Tastatur, auf das registrierte GUI-Element. Es erfolgt jedoch auch eine Kategorisierung der Listener nach der Art der Benutzeraktion, d.h. um einen Tastatureingabe abzufangen, wird beispielsweise für das Widget Text ein KeyListener registriert, welcher die Eingabe dann zur Verarbeitung weiterleitet.

Eclipse RCP baut auf SWT auf, siehe Abbildung 6, und verwendet weiterhin die höhere Schichten von JFace und der Forms API (seit Eclipse Version 3.0), welche SWT um Funktionalitäten ergänzen bzw. erweitern. Werden alle diese Betrachtungen zusammengefasst, ist es möglich zu sagen, dass RCP und somit auch kein Eclipse ohne SWT existieren können.

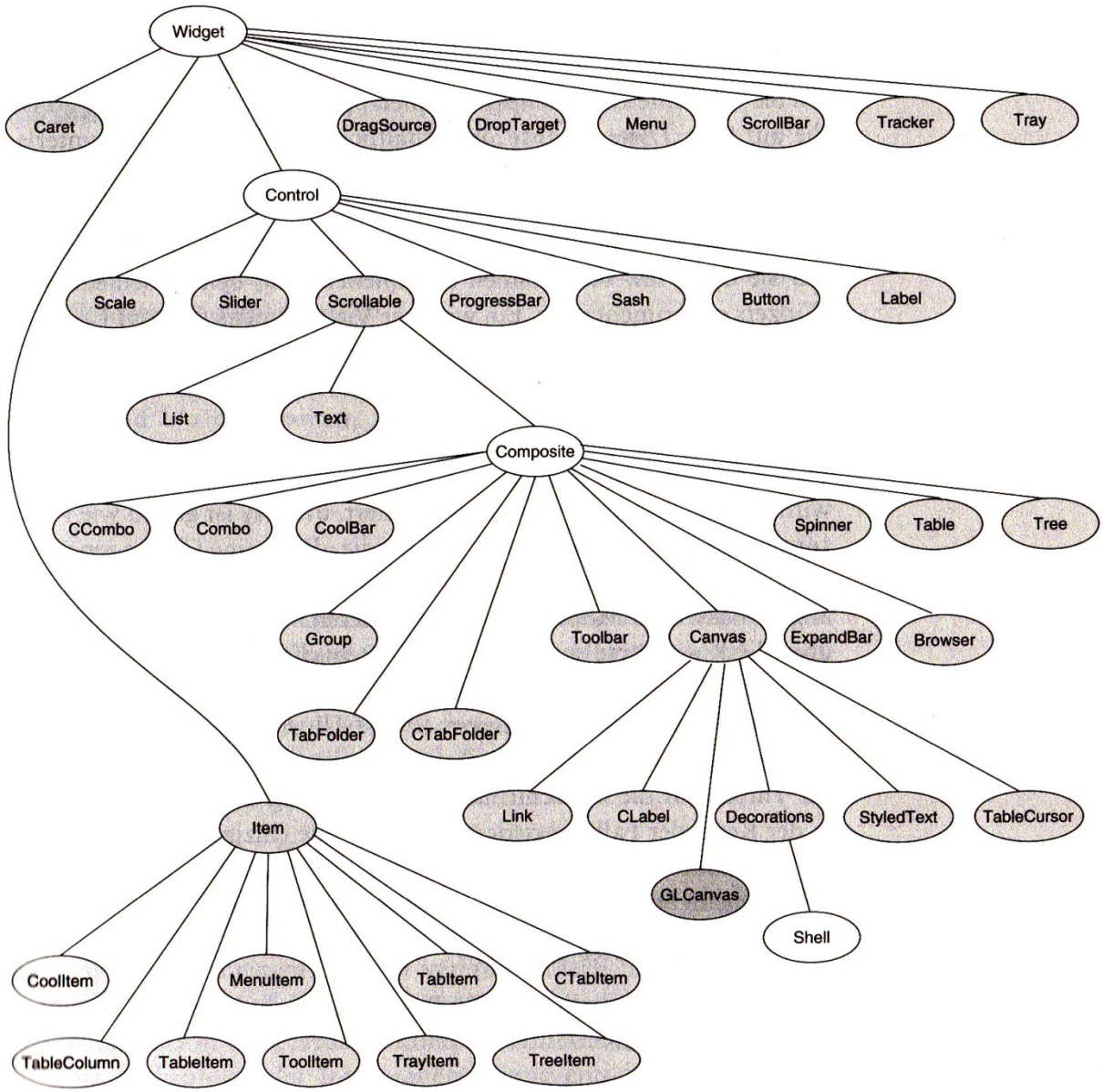


Abbildung 6: Widget-Hierarchie im SWT

Quelle: [Dau07]

3 Aufbau und Architektur der ALEA Commerce Suite

Die ALEA Commerce Suite besitzt ein mehrschichtiges Architekturmodell. Der eingesetzte Application-Server *Websphere* unterstützt dabei zwei verschiedene Plattformen für Hardware, welche in Abbildung 7 dargestellt sind. Entweder wird eine i-Series Maschine von IBM oder ein x-Series-Server von Intel verwendet. Bei Ersterem ist die Möglichkeit gegeben das native Betriebssystem der i5 zu verwenden, oder aber ein Linux. Die zweite Variante hingegen lässt nur ein Linux als Betriebssystem zu. Als Datenbank wird Database 2 (DB2) verwendet. Die Client-Anwendung kann beliebig auf einem Windows- oder Linux-Betriebssystem ablaufen, da dies aufgrund der Plattformunabhängigkeit von Java gewährleistet ist.

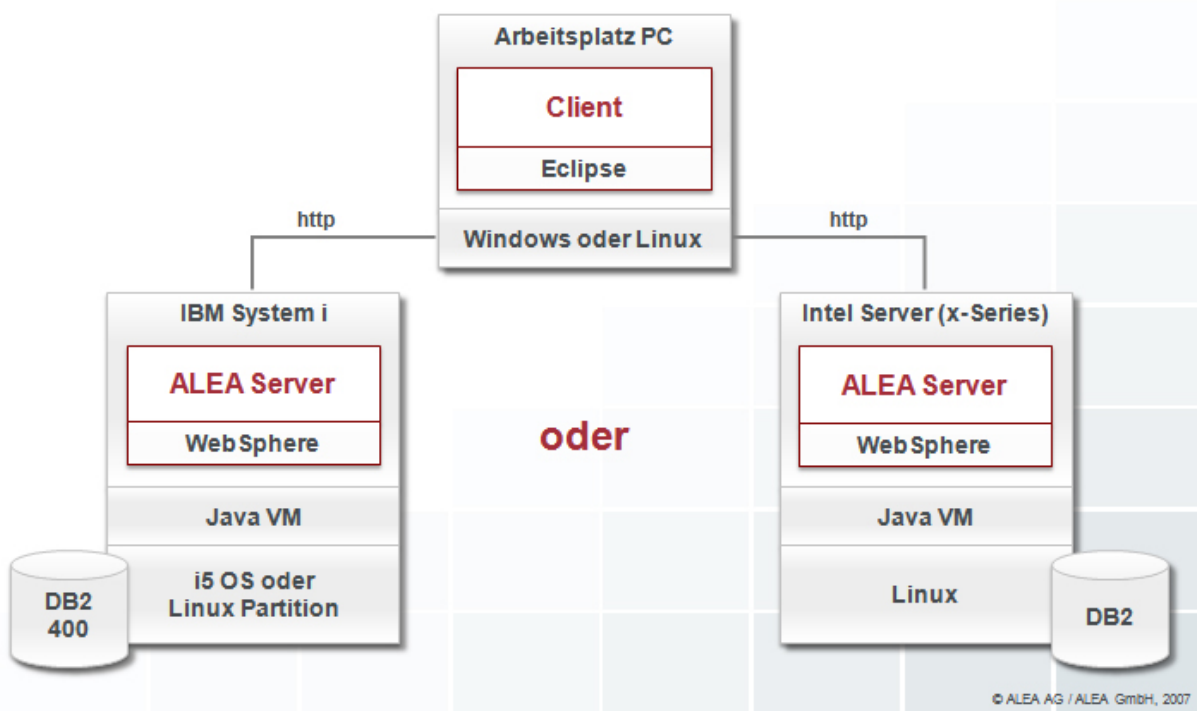


Abbildung 7: Architekturüberlick der ALEA Commerce Suite

Quelle: [Trö08], Folie 3

Der interne Aufbau der Anwendung besteht aus 4 Schichten; der Datenbankschicht (Database Layer), der Datenzugriffsschicht (Data Access Layer), der Geschäftslogikschicht (Business Layer) sowie der Präsentationsschicht (Presentation Layer), welche in Abbildung 8 illustriert sind. Da für die Lasttests bzw. das Sizing Funktionstests als Basis verwendet werden, erfolgen die Tests somit direkt gegen die Präsentationsschicht, also die grafische Benutzeroberfläche. Somit durchlaufen die Tests genau alle Schritte zur Abarbeitung der Funktion, wie der reale

Endanwender. Wie in Abbildung 8 zu sehen ist, hat der Client eine Verbindung mit dem Application Server[2], welcher wiederum eine zur verwendeten Datenbank DB2 besitzt. Die Verbindung zwischen Client und Application Server wird hierbei über das http-Protokoll realisiert, was in Abbildung 7 zu sehen ist. Der Client verwendet von den vier Schichten lediglich zwei, zum einen die Präsentationsschicht, da diese die grafische Oberfläche zur Verfügung stellt, zum anderen wird ein Teil der Geschäftslogikschicht zugunsten der Performance auf den Client ausgelagert. Der andere Teil dieser Schicht sowie auch die Datenzugriffsschicht liegen dagegen auf dem Application Server. Die Datenbankschicht der Anwendung ermöglicht die Kommunikation mit der Datenbank.

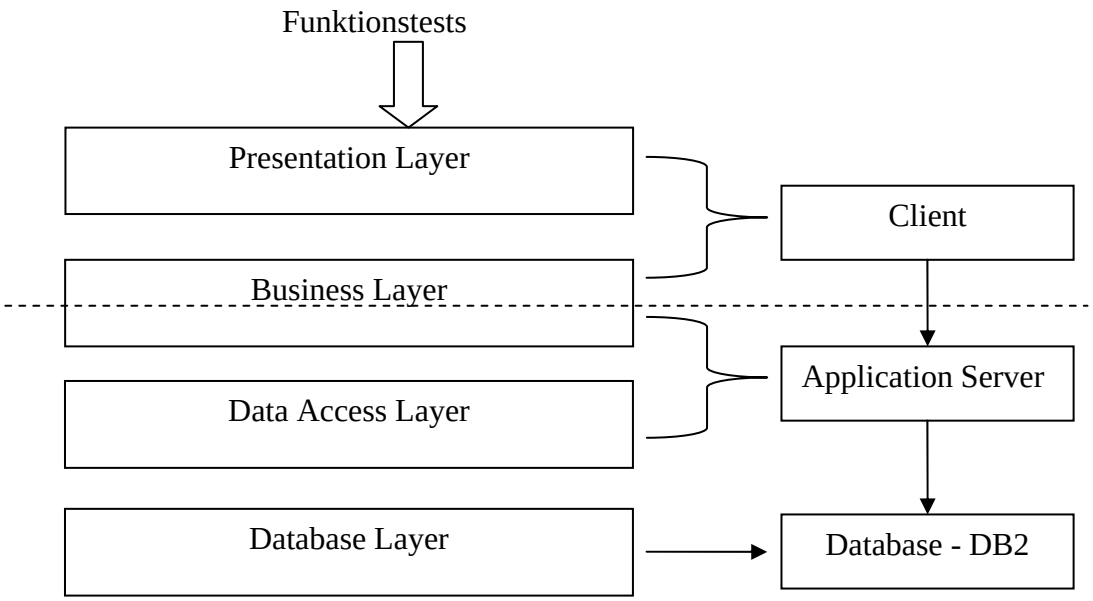


Abbildung 8: Die Schichten der ALEA Commerce Suite
Quelle: „vom Autor selbst erstellt“

[2] Dies ist ein Server für Anwendungsprogramme (Applikationen). In diesem Fall führt er Geschäftslogik der ALEA Commerce Suite aus, vermittelt zum Datenbankserver und erlaubt den Anschluss vieler Clients.

4 Evaluierung

4.1 Anforderungen

Um ein Tool zu finden, welches als GUI-Testautomatisierungswerkzeug für die ALEA Commerce Suite geeignet ist, müssen vorher die entsprechenden Anforderungen spezifiziert werden.
Da die ALEA Commerce Suite eine Rich-Client-Applikation ist und sich die grafische Oberfläche aus SWT-Elementen zusammensetzt, sind die ersten beiden Kriterien die Unterstützung für RCP und SWT in der Version 3.3. Des weiteren ist wichtig, dass Makros beziehungsweise Skripte einfach und schnell aufgezeichnet werden, um die Tests später in dieser Form oder aber auch modifiziert erneut aufzurufen zu können. Außerdem muss das Tool die Auswertungen ausgeführter Tests bereitstellen, zum Beispiel in Form von Reports oder Protokollen. Es muss also innerhalb dieser Auswertung ersichtlich sein, wann und an welcher Stelle im Test Fehler aufgetreten sind. Damit das Sizing mit dem Tool ausgeführt werden kann, muss eine Unterstützung für mehrere gleichzeitige Benutzer (Multi-User-Support) vorhanden sein. Das Kriterium, welches in der Entscheidungstabelle nicht enthalten ist, ist das der Zeit. Das gewählte Werkzeug muss sofort und ohne große Programmierarbeit verwendbar sein, da bereits ein Termin für einen Lasttest bzw. Sizing festgelegt ist.
Es gibt noch weitere Kriterien wie das Ausführen der Tests im Batch-Modus[3], das zentrale Monitoring der Clients in Bezug auf Antwortzeiten und die Unterstützung des http-Protokolls für eine spezielle Komponente der Anwendung, welche über http-Requests arbeitet. Die letztgenannten sind lediglich optionale Anforderungen, die das Werkzeug haben kann aber es nicht zwingend muss.

4.2 Zur Verfügung stehende Tools

Folgend werden die untersuchten Tools aufgeführt und eine Beschreibung zu vorhandenen Funktionalitäten gegeben. Bevorzugt wird ein Open Source Tool. Sollte jedoch keines dieser den Anforderungen genügen, so wird ein kommerzielles Werkzeug gewählt.

[3] In diesem Modus wird das Programm auf Kommandozeilenebene mit zusätzlichen Parametern gestartet. Der Ablauf bzw. die Abarbeitung geschieht vollkommen automatisch und ein manuelles Eingreifen ist nicht nötig.

4.2.1 Open Source Tools

4.2.1.1 Automated GUI Recorder

Der Automated GUI Recorder (AGR) ist Bestandteil der Eclipse Test & Performance Tools Platform (TPTP, http://www.eclipse.org/tptp/) und wird als solches auch weiterentwickelt. Wichtigste Merkmale des AGR sind der Support für SWT sowie RCP. Die anderen Kriterien können von diesem Tool nicht erfüllt werden. Da es Bestandteil eines ganzen größeren Projektes ist, wäre es zwar möglich dieses Tools so zu erweitern, dass es mit Hilfe anderer Plugins alle Anforderungen erfüllt, jedoch ist dabei ein Aufwand von mehren Wochen bzw. auch Monaten an Entwicklungsarbeit nötig. Da das gesuchte Tool schnellstmöglich zur Testautomatisierung und ohne große Programmierarbeit eingesetzt werden soll, kommt der Automated GUI Recorder daher als Kandidat nicht in Frage.

4.2.1.2 FIT

Das *Framework for Integrated Tests* (FIT, http://fit.c2.com) kann Funktionstests automatisieren. Dabei werden alle Funktionalitäten durch verschiedene Arten von HTML-Tabellen getestet. Wie der Begriff Framework aber schon deutlich macht, handelt sich hier nur um ein Grundgerüst. Zwar ist es möglich Skripte mit Hilfe von FIT zu erzeugen, aber die Test müssen alle manuell geschrieben, d.h. programmiert werden. In der Hinsicht auf die Zeit können aber keine Abstriche oder Kompromisse gemacht werden. Wie auch im Anhang A2 zu sehen ist, erfüllt FIT bis auf die Erstellung von Skripten keine der Anforderungen und scheidet somit als Tool zur Automatisierung aus.

4.2.1.3 The Grinder

The Grinder (http://grinder.sourceforge.net) ist ein Framework zur Automatisierung von Last-, Stress- und Funktionstests. Außer der Aufnahme der Aktionen eines echten Benutzers bietet *The Grinder* mehrere Möglichkeiten die Skripte benutzerdefiniert anzupassen, zum Beispiel in Form von Jython, mit dem ein Zugriff auf jede java-basierte API direkt im Skript gewährt wird. Als Skriptsprache kommt Python zum Einsatz. Ein Analyse- und Auswertungswerkzeug ist ebenfalls integriert. Hierbei wird u. a. die Anzahl der Aufrufe der Tests festgehalten und ob diese fehlgeschlagen sind oder erfolgreich abgearbeitet wurden.

Weiterhin werden das http-Protokoll, der Multi-User-Betrieb sowie der Batch-Modus unterstützt. Aufgrund des fehlenden Supports für die Rich-Client-Platform sowie für SWT kommt *The Grinder* nicht als potentieller Kandidat für das Testautomatisierungswerkzeug in Frage.

4.2.2 Kommerzielle Tools

4.2.2.1 Borland SilkPerformer

Das Werkzeug mit dem Namen *Silkperformer* der Firma Borland (http://www.borland.com) ist speziell für alle Arten von Performancetests konzipiert worden. Es bietet eine schnelle Erstellung von Skripten durch eine grafische Oberfläche sowie deren Bearbeitung. Hierbei wird eine Vielzahl von Protokollen unterstützt; genannt seien hier nur http(s) oder Java RMI (EJB/J2EE). Mit Hilfe der lasterzeugenden Agenten können 10 aber auch mehr als 10.000 Benutzer virtuell simuliert werden, welche das System unter Last setzen. Des weiteren gibt es weitgehende Möglichkeiten zur Auswertung und auch zur Echtzeit-Diagnose der lasterzeugenden Maschinen, besonders in Hinsicht von bestehender Last und daraus resultierenden Antwortzeiten. Hierbei werden die Client-PCs als auch Server betrachtet. Der *Silkperformer* ist für die Verwendung als Automatisierungstool für die ALEA Commerce Suite nicht geeignet, da mit dieser Software keine Java GUI-Objekte getestet werden können. Alle anderen Kriterien werden im umfangreichen Maß erfüllt.

4.2.2.2 Borland SilkTest

Borlands *SilkTest* setzt an genau der Stelle an, wo der *SilkPerformer* seine Grenzen hat, nämlich im Testen von GUI-Objekten. Es ist ein Werkzeug, das für automatisierte Tests von grafischen Oberflächen entwickelt wurde. Hierbei unterstützt es eine große Anzahl standardisierter Umgebungen wie beispielsweise Java GUIs (Eclipse 3.0 und 3.1, SWT 3.2 und RCP-Anwendungen), .NET GUIs (z.B. Visual Basic 6 bzw. Active X) und noch einige andere, siehe hierzu [Bor08b]. Nach Kontakt mit einem Mitarbeiter der Firma wurde eine Aktualisierung des Tools zum Testen von SWT-Elementen von Version 3.2 auf 3.3 auf ca. sechs bis sieben Wochen später festgelegt. Mit *SilkTest* können Tests durch die grafische Oberfläche schnell erstellt und auch entsprechend bearbeitet werden. Es besitzt eine Abstraktionsschicht, damit die Testskripts leicht anpassbar sowie wiederverwendbar sind.

Während des Tests erfolgt über die Technologie *TrueLog* die visuelle Diagnose der Aktionen, welche das Skript ausführt. Über Technologie, die Agent bereitstellt, welche auf verschiedenen Maschinen laufen, wird der Multi-User-Support sichergestellt. Ebenfalls unterstützt wird der Betrieb im Batch-Modus. Somit erfüllt Borlands *SilkTest* alle primären sowie optionalen Anforderungen bis auf das Echtzeit-Monitoring, wie auch im Anhang A2 zu erkennen ist. Es ist daher ein Kandidat für das Automatisierungswerkzeug.

4.2.2.3 GUIdancer

Der *GUIdancer*, welcher von der Firma Bredex GmbH (http://www.bredex.de) entwickelt wird, ist ein Testautomatisierungstool speziell für Java- und Webanwendungen. Da es selbst eine RCP-Anwendung ist, unterstützt es daher nativ RCP und SWT, aber auch HTML-Seiten können mit *GUIdancer* getestet werden. Die Spezifikation der Testfälle ist denkbar einfach, denn die Erstellung erfolgt interaktiv, Programmierkenntnisse sind nicht nötig. Das Besondere ist, dass die Testerstellung bereits vor der fertig erstellten Anwendung beginnt. Somit wachsen die Tests bereits mit der Softwareentwicklung. Es ist möglich, dass GUIdancer als eigenständige Anwendung oder sogar als Eclipse-Plugin läuft. Eine weitere Besonderheit ist die Robustheit gegen die zu testende Anwendung. Da die Spezifikation der Tests über ein Mapping erfolgt, welches objekt- und positionsabhängig ist, muss lediglich ein Neumapping, aber keine neue Spezifikation erfolgen, wenn die Anwendung geändert wird. Die Ergebnisse eines Tests werden in Form von html- oder xml-Dateien gespeichert. Sie enthalten alle zur Auswertung erforderlichen Daten, u. a. Name und Startzeit des Tests, Dauer und aufgetretene Fehler. Durch eine Client-Server-Struktur von *GUIdancer* ist es möglich, dass beide Teile verteilt installiert werden. So wird der Multi-User-Support bewerkstelligt. Für jede Instanz der zu testenden Anwendung muss hier ein Command Line Interface (CLI) gestartet werden. Über diese CLI wird auch der Batch-Betrieb möglich. Es ist lediglich kein Echtzeit-Monitoring integriert. Ebenso wie *SilkTest* von Borland erfüllt der GUIdancer alle Bedingungen als Automatisierungstool bis auf das Echtzeit-Montoring. Deswegen kommt diese Software als potentieller Kandidat in Frage.

4.2.2.4 HP Loadrunner

Wie auch *SilkPerformer* von Borland ist HPs *HP Loadrunner* (https//www.hp.com) ein Tool für Performancetests. So kann ein umfassender Bereich an Anwendungsumgebungen und Protokollen getestet werden, wie zum Beispiel Citrix, Java, .NET, Web- sowie alle großen ERP- und CRM-Anwendungen. Außerdem sind mehrere Anzeigen für das Echtzeit-Monitoring und Diagnosen der genannten Systeme verfügbar. HP Loadrunner integriert sich in die Entwicklungsumgebungen für J2EE, Microsoft .NET und Visual Studio von Microsoft. Die Testskripte lassen sich direkt in den genannten Entwicklungsumgebungen erstellen. Obwohl der *HP Loadrunner* eine große Anzahl an Umgebungen und Protokollen unterstützt, ist es mit diesem Produkt nicht möglich auf RCP- sowie SWT-basierende Anwendungen zu testen. Aus diesem Grund ist das Programm nicht in der engeren Auswahl für das Testautomatisierungswerkzeug.

4.2.2.5 HP QuickTest Professional

HPs *QuickTest Professional* ist ein Werkzeug zum automatisierten Testen von GUI-Objekten. Es unterstützt verschiedene Umgebungen und Technologien wie u. a. Eclipse 3.2, 3.3, SWT, Firefox 3.0, PeopleSoft 9.0, .NET 3.5. Über eine schlüsselwortgetriebene Technologie zum Testen von Anwendungen können Testskripte schnell erstellt und leicht gepflegt werden. Mit Hilfe einer speziellen Objektidentifizierungsmethode ist eine zuverlässige Testausführung gewährleistet, auch wenn die zu testende Applikation geändert wird. Unvorhersehbare Ereignisse der Anwendung während des Testens werden dabei durch den integrierten *Recovery Manager* behandelt. Obwohl *QuickTest Professional* den Support für RCP und SWT bereitstellt, als auch die Erstellung von Skripten ermöglicht, kommt es als Tool zur Automatisierung nicht in Frage, da die anderen gestellten Anforderungen wie beispielsweise Analyse- und Auswertungswerkzeuge oder der Multi-User-Support nicht erfüllt werden.

4.2.2.6 QALoad

Mit *QALoad* (http://www.compuware.com) der Compuware GmbH ist es möglich Performancetests durchzuführen. Um die Last für den Test zu erzeugen werden hunderte bzw. tausende von virtuellen Benutzern emuliert, welche dann die vorher erstellten Testskripte ausführen. Mit einem bestimmten Modul in QALoad ist die Möglichkeit gegeben, dass der

virtuelle Benutzer eine Rolle für die Anwendung zugewiesen bekommt und entsprechend dann Funktionstests dafür ausführt. Die Resultate dieser Tests können in Form von Berichten, Diagrammen oder Tabellen eingesehen werden. Der http-Support sowie der Batch-Modus werden ebenfalls unterstützt. Das Echtzeit-Monitoring wird jedoch nur mit einer extra Suite mit Namen *Vantage*, z.B. *ServerVantage*, hergestellt. Zwar wird Java an sich unterstützt, jedoch kein Testen von GUI-Objekten, weshalb *QALoad* als Werkzeug für die Automatisierung nicht in Frage kommt.

4.2.2.7 QF-Test

QF-Test (http://www.qfs.de/) ist ein Werkzeug, mit dem Funktionalitäts- als auch Lasttests für Java-Anwendungen mit einer grafischen Oberfläche erstellt werden können. Es wird von der Quality First Software GmbH entwickelt. Da die Software selbst in Java geschrieben ist, kann es sowohl auf Windows als auch auf Unix-Derivaten installiert werden. Je nach gewählter Edition des Programms wird entweder Swing oder SWT, in der Suite-Edition sogar beides unterstützt. Durch die grafische Oberfläche des Programms ist eine schnelle und intuitive Bedienung möglich, Tests können schnell und einfach erstellt werden. Daher sind Programmierkenntnisse nicht zwingend erforderlich. Für weiterführende, größere Test wird hier zusätzlich die integrierte Skriptsprache Jython zur Verfügung gestellt. Eine weitere wichtige Eigenschaft ist die Robustheit gegenüber der zu testenden Applikation, denn durch den verwendeten Wiedererkennungsmechanismus für die grafischen Objekte entsteht lediglich ein geringer Wartungsaufwand der erstellten Testskripte. Über den Bericht, der am Ende jeder Testsuite erzeugt wird, kann ermittelt werden, wie viele Tests ausgeführt und welche davon erfolgreich beendet wurden und welche fehlgeschlagen sind. Die Berichte werden als xml- und als html-Datei bereitgestellt. Außerdem werden Angaben zur benötigten Zeit als auch zum verwendeten System gemacht. Bis auf das Echtzeit-Monitoring erfüllt QF-Test alle geforderten Kriterien, siehe Anhang A2, und ist deshalb ein Kandidat für das Automatisierungstool.

4.2.2.8 Squish for Java

Mit diesem Produkt der froglogic GmbH (http://www.froglogic.com) gibt es ein weiteres Tool, welches speziell zum Testen von Java GUI-Objekten konzipiert wurde. Unterstützt

werden AWT/Swing, SWT sowie RCP-Anwendungen. Für die Erstellung von Skripten wird dem Benutzer freie Auswahl zwischen vier verschiedenen Skriptsprachen gelassen: Python, JavaScript, Perl und Tcl. Es werden bei der Testerstellung alle einfachen und komplexen Java GUI-Objekte erkannt, außerdem auch selbst erstellte. Dieser Erkennungsmechanismus ist sehr robust gegenüber der Applikation und so funktionieren Tests auch, wenn die Anwendung geändert wurde. Die zu testende Anwendung kann anschließend von verschiedenen Maschinen und Plattformen aus getestet werden, was von einem zentralen Punkt aus steuerbar ist. Über die verfügbare Kommandozeile ist es weiterhin möglich *Squish for Java* im Batch-Modus zu betreiben. Der http-Support ist aber nicht nativ enthalten, sondern es wäre die Lizensierung der Edition *Squish for Web* erforderlich. Ebenfalls ist keine Echtzeit-Monitoring enthalten. Aufgrund der fehlenden Anforderung nach Analyse- und Auswertungswerkzeugen kommt dieses Produkt nicht in die engere Auswahl.

4.3 Entscheidung

Da nicht alle Funktionalitäten in den sogenannten *Data Sheets* (Datenblättern) ersichtlich waren, war es nötig mit den Mitarbeitern der jeweiligen Firmen in Kontakt zu treten, um die noch benötigten Informationen zu erhalten. Aus den zur Verfügung stehenden Tools fiel die Entscheidung am Schluss auf QF-Test von Quality First Software. Alle geforderten primären Kriterien werden durch das Tool erfüllt. Außerdem sind die Lizenzpreise im Vergleich zu *GUIdancer* und *SilkTest* wesentlich günstiger.

Borlands *SilkTest* bot zwar eine Aktualisierung des SWT-Supports auf Version 3.3 an, jedoch erst nach mehreren Wochen. Da die Evaluierung und Testerstellung sowie das Sizing einem Zeitrahmen unterlagen, war dieses Tool letztendlich deshalb nicht geeignet. Der GUIdancer kam schließlich auch nicht in Frage, da einerseits die Lizenzpreise aus Sicht der Firma ALEA GmbH zu teuer waren, andererseits enthält das Tool ein Feature, welches zur Testautomatisierung für die in der Firma hergestellte Software nicht benötigt wird. Die Entscheidungstabelle mit den Tools und den geforderten Kriterien ist im Anhang A2 zu finden.

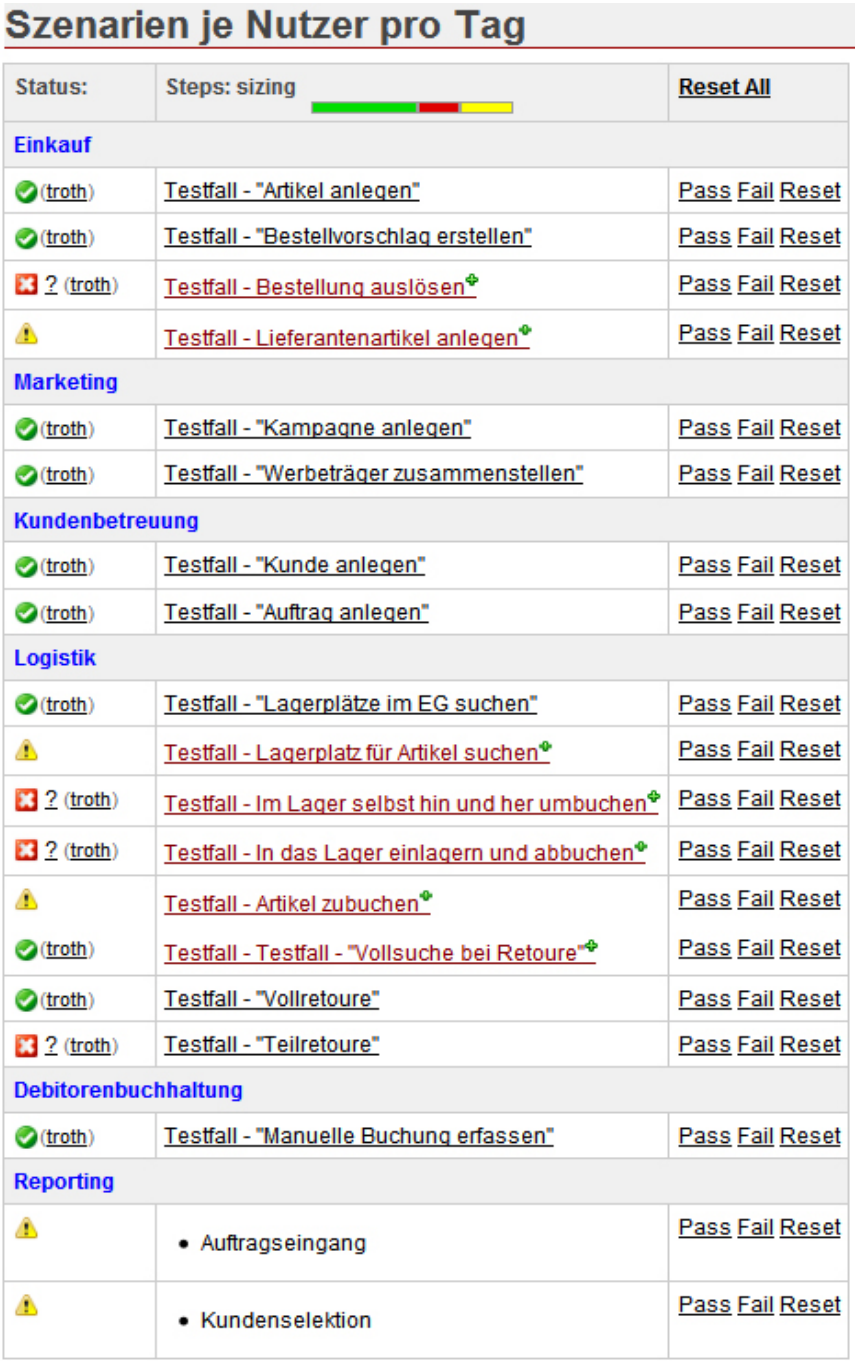
5 Einführung des Tools

In diesem Abschnitt werden die Implementierung des Tools bei der ALEA GmbH für die Funktionstests sowie das Sizing beschrieben.

5.1 Testvorbereitung

5.1.1 Geschäftsvorfälle

Damit der Lasttest respektive das Sizing durchgeführt werden kann, müssen die Tests automatisiert werden. Wie aus Abbildung 3 zu entnehmen ist, müssen deshalb zunächst die Geschäftsvorfälle festgehalten und diese Aspekte in Testskripte umgewandelt werden. Eine typische Auswahl von durchgeführten Aktionen im Versandhandel ist in Abbildung 9 zu finden. Hierbei ist zu beachten, dass die Anzahl der Geschäftsvorfälle pro Tag und Kunde variieren, was ebenso Auswirkungen im Lasttest nach sich zieht, da bestimmte Funktionen dann mehr genutzt werden als andere. Zum Beispiel werden die Geschäftsvorfälle *Auftrag anlegen* und *Kunde anlegen* öfters verwendet als beispielsweise *Vollretoure*.



Szenarien je Nutzer pro Tag

Status:	Steps: sizing	Reset All
Einkauf		
(troth)	Testfall - "Artikel anlegen"	Pass Fail Reset
(troth)	Testfall - "Bestellvorschlag erstellen"	Pass Fail Reset
? (troth)	Testfall - Bestellung auslösen	Pass Fail Reset
	Testfall - Lieferantenartikel anlegen	Pass Fail Reset
Marketing		
(troth)	Testfall - "Kampagne anlegen"	Pass Fail Reset
(troth)	Testfall - "Werbeträger zusammenstellen"	Pass Fail Reset
Kundenbetreuung		
(troth)	Testfall - "Kunde anlegen"	Pass Fail Reset
(troth)	Testfall - "Auftrag anlegen"	Pass Fail Reset
Logistik		
(troth)	Testfall - "Lagerplätze im EG suchen"	Pass Fail Reset
	Testfall - Lagerplatz für Artikel suchen	Pass Fail Reset
? (troth)	Testfall - Im Lager selbst hin und her umbuchen	Pass Fail Reset
? (troth)	Testfall - In das Lager einlagern und abbuchen	Pass Fail Reset
	Testfall - Artikel zubuchen	Pass Fail Reset
(troth)	Testfall - Testfall - "Vollsuche bei Retoure"	Pass Fail Reset
(troth)	Testfall - "Vollretoure"	Pass Fail Reset
? (troth)	Testfall - "Teilretoure"	Pass Fail Reset
Debitorenbuchhaltung		
(troth)	Testfall - "Manuelle Buchung erfassen"	Pass Fail Reset
Reporting		
	• Auftragseingang	Pass Fail Reset
	• Kundenselektion	Pass Fail Reset

Abbildung 9: Typische Geschäftsvorfälle im Bereich Versandhandel

Quelle: „vom Autor erstellt“

5.1.2 Skripterstellung

Bevor mit der Erstellung der Skripte begonnen werden kann, musst QF-Test zunächst eingerichtet werden. Die Systemeinstellungen werden in der Datei *qftest.cfg* gespeichert. Da jedoch diese Einstellungen nicht nur bei diesem einen Rechner benötigt werden, sondern für

den Lasttest auch an mehreren, wurden die Einstellungen in die Datei *alea.cfg* geschrieben, welche beim Start des Programms im Batch-Modus als Parameter übergeben wird, siehe auch Abschnitt 5.2.2. Dies ist notwendig, da der Lasttest beziehungsweise das Sizing ausschließlich im Batch-Modus abläuft.

In Abbildung 10 ist QF-Test im interaktiven, also grafischen Modus gestartet worden. Die Oberfläche ist klar strukturiert und übersichtlich gehalten. Sie setzt sich dabei aus vier Komponenten zusammen: der Menü- und Aktionsleiste, worüber zum Beispiel Tests aufgezeichnet und wiedergegeben werden können, der hierarchischen Baumansicht zum Navigieren innerhalb der geöffneten Testsuite, dem kontextabhängigen Editor zum Bearbeiten der Optionen der einzelnen Testschritte sowie dem Terminal zur sofortigen Anzeige von Fehlern, beispielsweise bei einem fehlschlagenden Start der zu testenden Applikation.

QF-Test ist ein Capture/Replay-Werkzeug, d.h. es wird eine Aufnahme gestartet und wie bei einem manuellen Test mit der Anwendung interagiert. Im Hintergrund zeichnet QF-Test nun die Aktionen bzw. Eingaben des Benutzers auf. Nach Abschluss der Aufnahme kann der Test nun automatisch nochmals wiedergegeben werden. Im Gegensatz zu anderen Capture/Replay-Werkzeugen merkt sich QF-Test nicht die Position der zu testenden Komponenten in Form von Pixeln am Bildschirm, sondern die Komponenten werden durch sogenannte Identifier (IDs) erkannt. So arbeitet der Wiedererkennungsmechanismus des Programms. Damit ist die Robustheit gegenüber der Anwendung gewährleistet und die Tests funktionieren auch nach Änderungen in der Software weiterhin.

Als Beispiel soll der Testfall *Kunde anlegen* der Testsuite dienen, welche in Abbildung 10 aufgerufen wurde. Für jeden Testfall gibt es die beiden Schritte Vorbereitung und Aufräumen. Damit wird sichergestellt, dass QF-Test die Anwendung korrekt gestartet und am Schluss diese auch wieder geschlossen hat. Das Schließen der zu testenden Anwendung ist nötig, weil so Fehler zwischen einzelnen Testfällen verhindert werden. Nach der Vorbereitung beginnt nun der Test der eigentlichen Funktion des Programms. In den folgenden Schritten gibt es eine ganze Reihe von Tastatureingaben oder Mausklicks auf der Oberfläche der Anwendung, die zu einer Sequenz zusammengefasst sind. Durch die Aufzeichnung sind den Tastatur-eingaben feste Werte zugewiesen worden, zum Beispiel bei der Eingabe des Vor- und Nachnamens des angelegten Kunden. Über den Editor besteht aber nun nachträglich die Möglichkeit diese Eingaben variabel zu machen, sodass hier Zufallsdaten aus einer

Datenbank verwendet werden können. Sind alle gewünschten Änderungen vorgenommen, kann jetzt der Test zur Überprüfung des fehlerfreien Ablaufes gestartet werden.

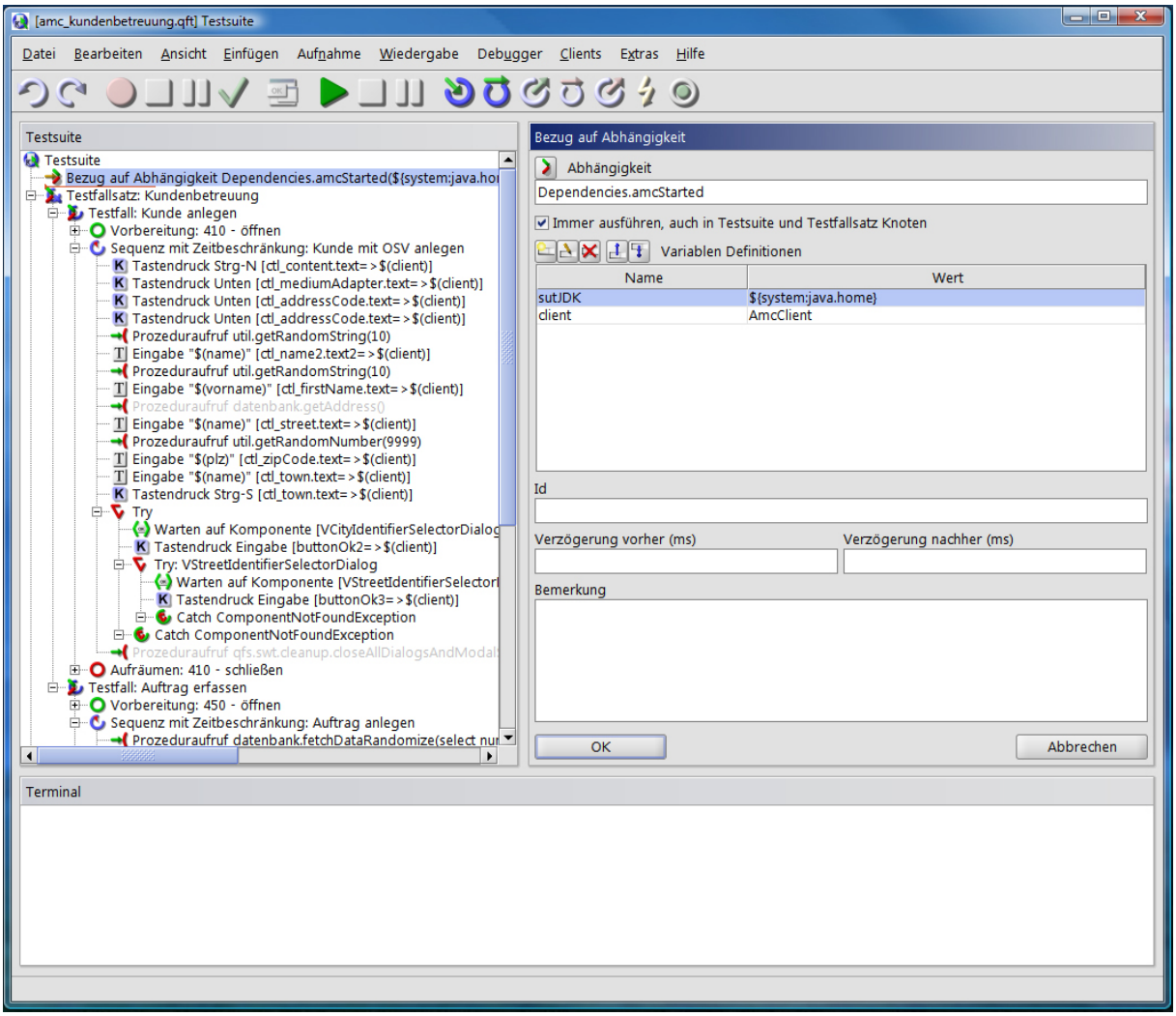


Abbildung 10: Grafische Oberfläche von QF-Test

Quelle: „vom Autor erstellt“

5.1.3 Standard-Funktionstests

Die Standard-Funktionstests sind diejenigen, welche bei der ALEA GmbH die grundlegenden Funktionen der ALEA Commerce Suite nach einem erfolgreichen Neubau der Anwendung mit allen bis dato gemachten Änderungen im Programm testen. Weiterhin gibt es eine Unterscheidung zwischen kurzem und langem Test nach dem Neubau. Ersterer wird verwendet um lediglich die Grundfunktionalitäten zu überprüfen, also der generelle Funktionstest. Die folgend genannten Testfälle charakterisieren ihn:

- Content Location prüfen
- Kampagne anlegen
- Werbeträger anlegen
- Artikel anlegen
- Artikel einem Werbeträger zuordnen
- Kunde anlegen
- Auftrag anlegen
- Lieferkontrolle für neue Aufträge ausführen
- Sendungen abrufen und Picklisten erstellen
- Uniform Resource Locator (URL) zum Data Warehouse stimmt
- Data Warehouse ist gefüllt

Dagegen wird der lange Test vor der Auslieferung einer neuen Version durchgeführt.
Für die o. g. Testfälle wurden dementsprechend Testskripte erstellt, falls dies nicht schon für das Sizing erfolgte.

5.2 Testautomatisierung

5.2.1 Allgemein

Für die Automatisierung wurden mit Hilfe von QF-Test für die Bereiche des Versandhandels Debitorenbuchhaltung, Einkauf, Marketing, Kundenbetreuung sowie Logistik Testsuiten erstellt, welche die jeweiligen in Abbildung 9 dargestellten Testfälle beinhalten. Außerdem werden diese einzelnen Testsuiten in einer Master-Testsuite aufgerufen, welche schließlich zu Beginn des Sizings ausgeführt wird. Die für die Automatisierung verwendete Infrastruktur wurde dabei von den Mitarbeitern der ALEA GmbH bereitgestellt.

5.2.2 Der Batch-Modus

Sind die Tests fertig erstellt und laufen im interaktiven (grafischen) Modus ohne Fehler und möglichst ohne Ausnahmen, so kann nun das Programm im Batch-Modus aufgerufen werden. Dies geschieht über die Kommandozeile bzw. über eine Batch-Datei, in der der Befehl zum Aufruf von QF-Test und zugehörige Parameter enthalten sind. Über den Parameter *–batch* wird anstelle der grafischen Oberfläche der Test auf der Kommandozeile abgearbeitet. Ein

Eingreifen ist somit hier nicht mehr nötig. Mit *–systemcfg <Dateiname>* werden die zu verwendenden Systemeinstellungen geladen. Normalerweise sind diese in der qftest.cfg enthalten, jedoch wird eine abgeänderte Variante benötigt und die Änderungen wurden in die Datei *alea.cfg* geschrieben. Durch die Option *–runid <Platzhalter>* wird dem Testlauf eine ID zugewiesen. Diese wiederum setzt sich aus beliebigen Platzhaltern zusammen. In diesem Fall wären das +M, +d, +H und +m. So setzt sich die ID also aus dem aktuellen Monat, Tag, der aktuellen Stunde sowie Minute zusammen. Bei Aufruf von QF-Test mit dem Parameter *–threads <Anzahl>* können parallel mehrere Threads gestartet werden, welche die angegebene Testsuite ausführen. Dafür werden aber je nach Anzahl der angegebenen Threads Runtime-Lizenzen, also Lizenzen zum Ausführen von Testsuiten, benötigt. Weiterhin wird, wie in Abbildung 11 zu sehen, die Option *-runlog <Verzeichnis>* angegeben. So wird für die ausgeführte Testsuite am Ende ein Protokoll im Verzeichnis *logs* angelegt. Der Platzhalter */+i* bezeichnet hierbei die vorher erstellte Run-ID. Folglich befindet sich im genannten Verzeichnis dann zum Beispiel ein Protokoll *08041703.qrl*. Der Bericht über den Verlauf der Testsuite wird mit Hilfe des Parameters *–report <Verzeichnis>* erstellt. Wie in Abbildung 11 illustriert, wird als Verzeichnisname *report_+i* verwendet, wobei *+i* wie auch bei den Protokollen die aktuelle Run-ID bezeichnet. Zum Schluss des Befehls wird noch der Pfad und Dateiname der auszuführenden Testsuite in Anführungszeichen angegeben.

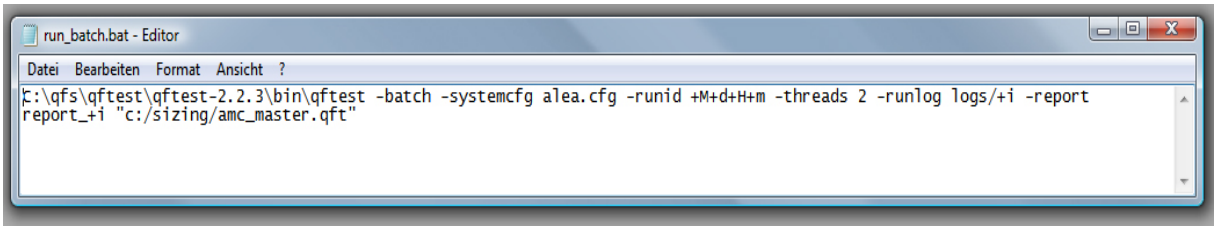


Abbildung 11: Datei zum Aufruf von QF-Test im Batch-Modus

Quelle: „vom Autor erstellt“

5.3 Hardware-Sizing

Wie aus Abbildung 12 ersichtlich gibt es für das Sizing zunächst eine Einteilung nach der Größe des Kunden, also seiner Mitarbeiter in den jeweiligen Bereichen. Daraus und aus anderen Informationen wie zum Beispiel der maximalen Anzahl an Aufträgen pro Tag kann nun eingestellt werden, wie oft welche Testsuite und darin enthalten welcher Testfall aufgerufen wird. Über ein Jython-Skript, welches in QF-Test eingebaut ist, wird jedes Mal

über einen Zufallsgenerator einer der Testfälle aus den Testsuiten aufgerufen. Für jeden Testfall wurde dabei ein Zahlenbereich festgelegt. Daraus ergibt sich, dass je größer der Zahlenbereich für einen Testfall gewählt wird, desto größer ist die Wahrscheinlichkeit, dass er ausgeführt wird. Durch die Anpassung der Zahlenbereiche in diesem Skript können also die die verschiedenen Bereiche intensiver getestet werden, je nachdem wie dies in der Realität bei dem Kunden der Fall ist.

Während des Sizings sollen weiterhin die Antwortzeiten von Seiten des Clients gemessen werden. Als Kriterium wurde festgelegt, dass die Zeitspanne für den Wechsel zwischen zwei Eingabe-/Aktionsfeldern maximal eine halbe Sekunde betragen darf. Sollte sie dennoch länger dauern, dann muss die gesamte Antwortzeit des Testfalls kleiner oder gleich aller Wechsel multipliziert mit der Antwortzeit von einer halben Sekunde sein. Da jedoch QF-Test das Echtzeit-Monitoring nicht ermöglicht, wird dies mit einem internen Tool des verwendeten Servers, also mit einem i5-eigenen Monitoring Tool durchgeführt.

Der Testplan sieht vor, dass immer das mögliche Maximum getestet wird. In bestimmten Zeiten werden zum Beispiel 1000 Aufträge pro Tag anstatt von ca. 400 entgegengenommen. Dabei muss aber beachtet werden, dass sich diese Aufträge nicht gleichmäßig über den Tag verteilen, sondern ungefähr die Hälfte dessen in einem Zeitraum von vier Stunden angelegt wird, was ca. 125 Aufträge pro Stunde sind. Dies muss das System leistungsmäßig ermöglichen.

Kunde

Mitarbeiter	Kundengruppe A	Kundengruppe B	Kundengruppe C
Anzahl Kundenservice/ Auftragsverwaltung	20	40	200
Anzahl Debitorenverwaltung	3	4	20
Anzahl Einkauf	5	8	30
Anzahl Marketing	2	10	20
Anzahl Lager	5	10	30
Anzahl Controlling	2	2	10
Anzahl Diverses	2	10	1

Abbildung 12: Ausschnitt aus den Anforderungen für das Sizing

Quelle: „vom Autor erstellt“

5.4 Auswertung und Dokumentation

5.4.1 Auswertung

5.4.1.1 Das Protokoll

Ist eine Testsuite komplett abgearbeitet worden, so wird anschließend aus den Ergebnissen ein Protokoll generiert, welches vor allem zur Fehlersuche geeignet ist, da hier jeder Schritt eines Tests genau festgehalten ist, wie in Abbildung 13 zu sehen. Sind während der Durchführung Ausnahmen oder gar Fehler aufgetreten, so sind von Beginn an im Protokoll in der Baumansicht rote Vierecke zu erkennen. Dadurch ist sehr schnell erkennbar, an welcher Stelle im Testfall das Problem aufgetreten ist. Man kann sich entweder direkt über die Baumansicht bis zum Fehler klicken oder verwendet die entsprechende Tastenkombination. Um die Fehlerfindung in dem Skript zu vereinfachen, enthält das Protokoll zusätzlich noch ein Bildschirmabbild zu dem Zeitpunkt, als der Fehler aufgetreten ist.

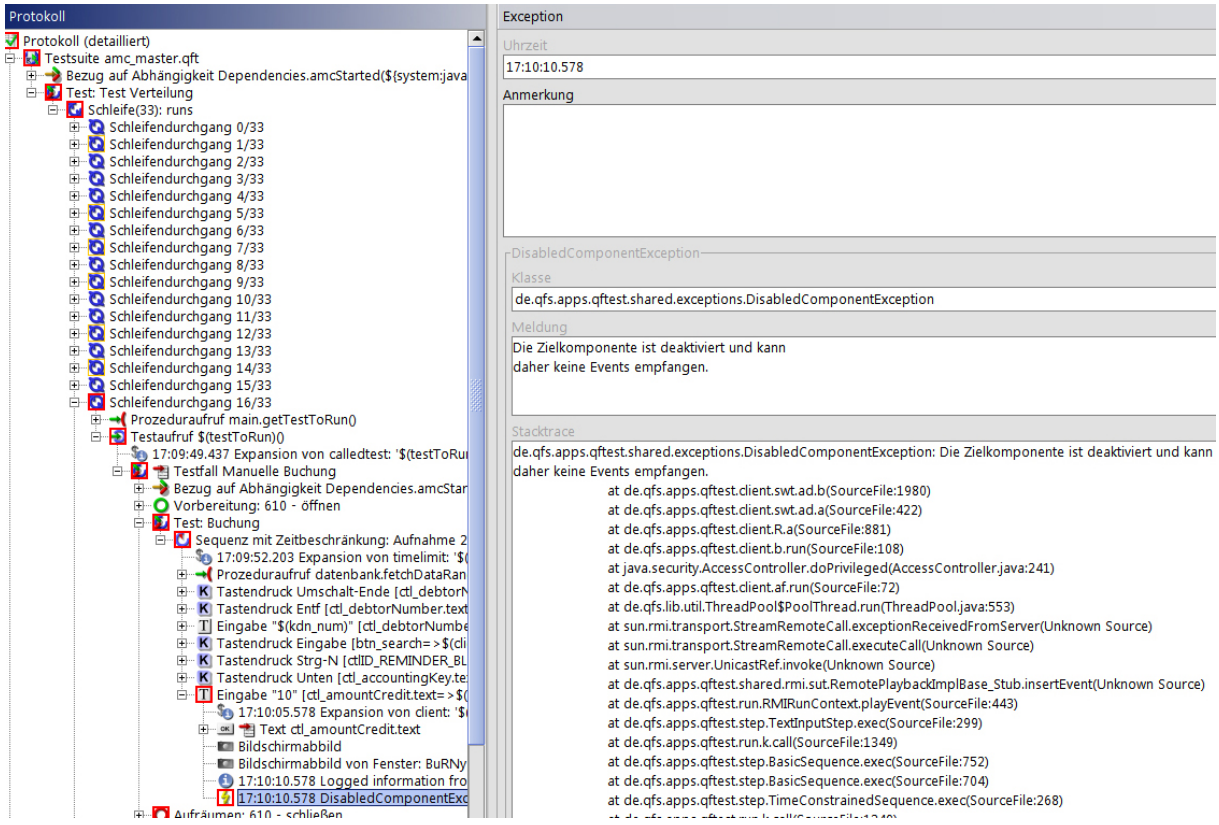


Abbildung 13: Ausschnitt aus einem von QF-Test erstellten Protokoll

Quelle: „vom Autor erstellt“

5.4.1.2 Der Bericht

Es ist jetzt möglich aus dem vorher generierten Protokoll einen Bericht zu erstellen. Der Bericht kann wahlweise im html- oder xml-Format, aber auch, wenn gewünscht, in beiden Formaten erzeugt werden. Weiterhin gibt es Einstellmöglichkeiten, was in dem Bericht alles angezeigt werden soll, also zum Beispiel Fehler, Ausnahmen usw. Entsprechend können über eine weitere Option Bildschirmabbilder aus dem Protokoll ebenfalls in den Bericht übernommen und skaliert werden.

In Abbildung 14 ist der erste Teil eines Berichtes für eine durchgeführte Testsuite zu sehen. Zu Beginn gibt es eine allgemeine Zusammenfassung wichtiger Daten wie z. B. der Name des ausgeführten Tests, das verwendete Betriebssystem, welche Java- und QF-Test-Versionen verwendet wurden, sowie Dauer und aufgetretene Fehler während des Testdurchlaufs. In der rechten oberen Ecke gibt eine Legende Aufschluss über die verwendeten Symbole und ihre Bedeutung im Bericht. In dem Gesamtergebnis werden nun explizit die Anzahl von Fehlern und Ausnahmen angezeigt, aber auch andere wichtige Informationen wie die zur benötigten Zeit für den Testdurchlauf oder die prozentuale Anzeige erfolgreich abgeschlossener Testfälle.

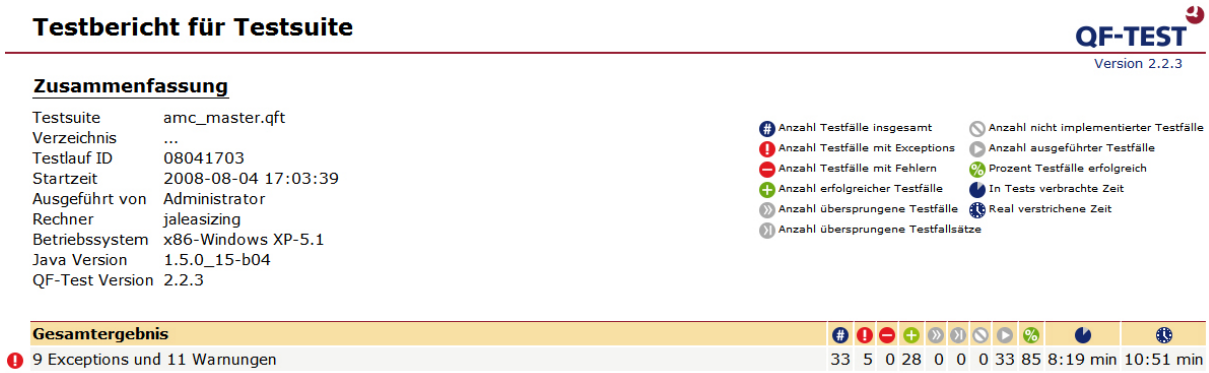
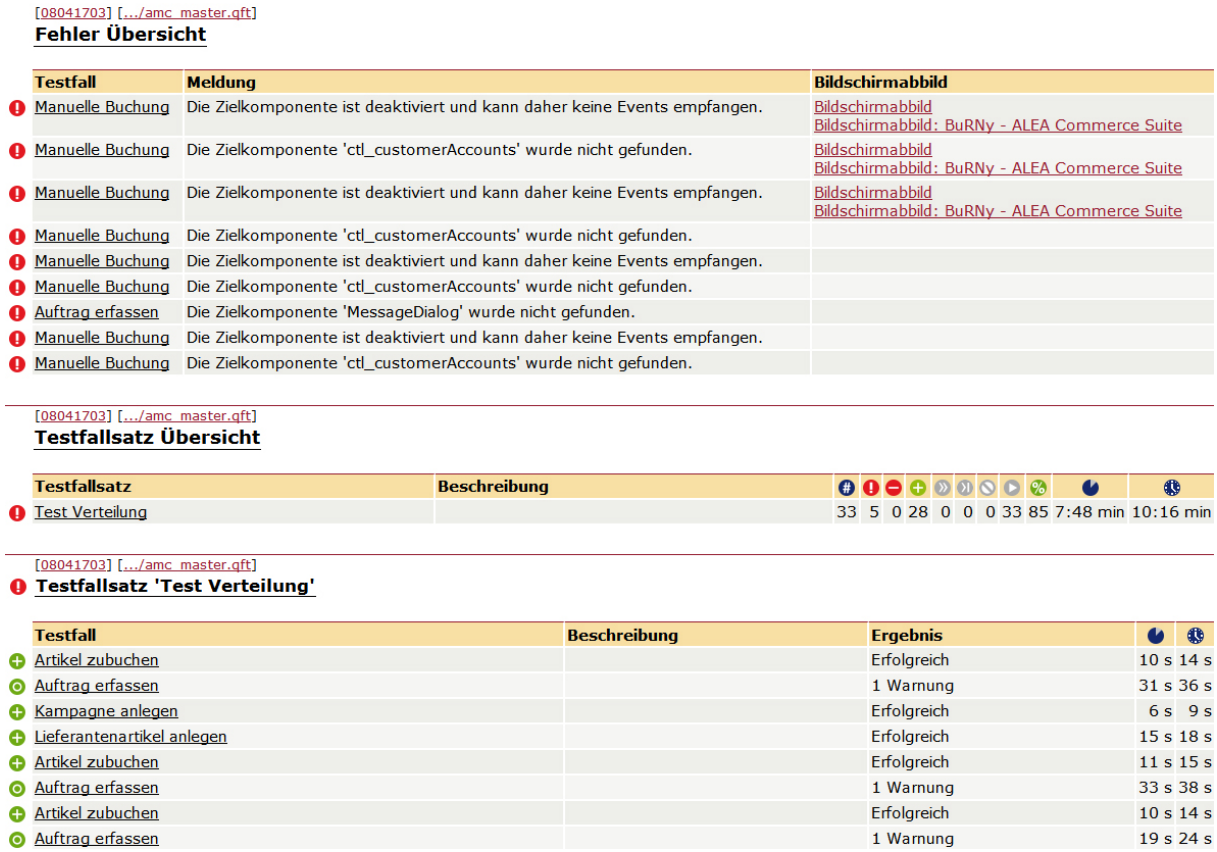


Abbildung 14: Allgemeine Zusammenfassung des Testberichtes

Quelle: „vom Autor erstellt“

Anschließend werden nun die in den Testfällen aufgetretenen Fehler und Ausnahmen aufgeführt. Wurde bei der Generierung des Berichtes die zuvor genannte Option für die Einbettung der Bildschirmabbilder aktiviert, so sind diese nun hier verfügbar, siehe Abbildung 15. Danach werden die vorhandenen Testfallsätze angezeigt; hier in diesem Fall lediglich einer. Testfallsätze sind dafür gedacht, dass Tests in einer Testsuite strukturiert

werden können. Innerhalb des Testfallsatzes erfolgt eine Auflistung der enthaltenen Testfälle, deren Ergebnis und Dauer.



[08041703] [.../amc_master.qft]

Fehler Übersicht

Testfall	Meldung	Bildschirmabbild
Manuelle Buchung	Die Zielkomponente ist deaktiviert und kann daher keine Events empfangen.	Bildschirmabbild Bildschirmabbild: BuRNy - ALEA Commerce Suite
Manuelle Buchung	Die Zielkomponente 'ctl_customerAccounts' wurde nicht gefunden.	Bildschirmabbild Bildschirmabbild: BuRNy - ALEA Commerce Suite
Manuelle Buchung	Die Zielkomponente ist deaktiviert und kann daher keine Events empfangen.	Bildschirmabbild Bildschirmabbild: BuRNy - ALEA Commerce Suite
Manuelle Buchung	Die Zielkomponente 'ctl_customerAccounts' wurde nicht gefunden.	
Manuelle Buchung	Die Zielkomponente ist deaktiviert und kann daher keine Events empfangen.	
Manuelle Buchung	Die Zielkomponente 'ctl_customerAccounts' wurde nicht gefunden.	
Auftrag erfassen	Die Zielkomponente 'MessageDialog' wurde nicht gefunden.	
Manuelle Buchung	Die Zielkomponente ist deaktiviert und kann daher keine Events empfangen.	
Manuelle Buchung	Die Zielkomponente 'ctl_customerAccounts' wurde nicht gefunden.	

[08041703] [.../amc_master.qft]

Testfallsatz Übersicht

Testfallsatz	Beschreibung	#	!	−	+	»	»	⊘	▶	%		
Test Verteilung		33	5	0	28	0	0	0	33	85	7:48 min	10:16 min

[08041703] [.../amc_master.qft]

Testfallsatz 'Test Verteilung'

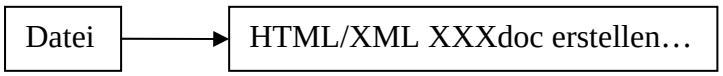
Testfall	Beschreibung	Ergebnis		
Artikel zubuchen		Erfolgreich	10 s	14 s
Auftrag erfassen		1 Warnung	31 s	36 s
Kampagne anlegen		Erfolgreich	6 s	9 s
Lieferantenartikel anlegen		Erfolgreich	15 s	18 s
Artikel zubuchen		Erfolgreich	11 s	15 s
Auftrag erfassen		1 Warnung	33 s	38 s
Artikel zubuchen		Erfolgreich	10 s	14 s
Auftrag erfassen		1 Warnung	19 s	24 s

Abbildung 15: Übersicht zu Fehlern und den Testfällen

Quelle: „vom Autor erstellt“

5.4.2 Dokumentation

Dokumentationen können für in QF-Test vorhandenen Testfallsätzen, Testfällen, Paketen und Prozeduren erstellt werden. Diese können sowohl im grafischen, als auch im Batch-Modus erfolgen. Im grafischen Modus erfolgt dies bei einer geöffneten Testsuite über die Menüleiste



Wie auch zuvor im Unterpunkt 5.4.1.2 ist hier eine Auswahl als html- und/oder xml-Datei möglich. Der Eintrag *Testdoc* gilt für Testfallsätze und Testfälle, die Option *Pkgdoc* hingegen wird für erstellte Pakete und Prozeduren verwendet. Im Batch-Modus wird dies über die folgenden, zusätzlich zum Programmaufruf gehörenden Parameter realisiert:

```
qftest –batch -gendoc –testdoc <Zielverzeichnis> <Verzeichnis mit den Testsuiten>
```

```
qftest –batch -gendoc –pkgdoc  <Zielverzeichnis> <Verzeichnis mit den Testsuiten>
```

Es ist auch möglich beides gemeinsam zu erstellen. Bei der Erstellung der Dokumentationen wird das Feld *Bemerkungen* verwendet, siehe Abbildung 10. Es können hier html-Kennzeichnungen als auch die aus JavaDoc bekannten Doctags, welche immer mit einem @ beginnen, verwendet werden. JavaDoc ist ein Format zur Dokumentation von Java-Programmen. Über @author kann zum Beispiel der Autor des Testfalls festgelegt werden, mit @since, seit wann dieser Test mit in die Automatisierung aufgenommen wurde. Welche weiteren Doctags verwendet werden können, sind entsprechend dem Handbuch von QF-Test zu entnehmen.

6 Fazit

Durch die Automatisierung der Tests kann jetzt die ALEA Commerce Suite nach einem Neubau mit allen durchgeführten Änderungen mit Hilfe von QF-Test nach Fehlern überprüft werden. So kann der kurze Eingangstest mit den wichtigen Funktionen schnell durchgeführt werden. Die Tester müssen sich dann lediglich noch auf die Tests konzentrieren, welche wegen der zu hohen Anpassungskosten nicht automatisiert werden können.

Sowohl für die Standard-Funktionstests als auch für das Sizing wurden von den Testfällen Skripte mit QF-Test erstellt. Das Sizing lief im August bereits einmal erfolgreich ab.

Literaturverzeichnis

[Ade08] Adersberger, Josef:
Wiki-getriebene Akzeptanztests,
http://it-republik.de/jaxenter/artikel/Wiki-getriebene-Akzeptanztests-0787.html,
Abruf: 2008.08.26

[AF08] Aston, Philip; Fitzgerald, Philip:
Features of The Grinder 3,
http://grinder.sourceforge.net/g3/features.html,
Abruf: 2008.04.02

[BD08] Beyer, Karsten; Dietrich, Dennis:
Methoden und Werkzeuge der Softwareproduktion
Stress- und Lasttest,
swt.cs.tu-berlin.de/lehre/mwsp/ws0304/ausarbeitungen/StressLast.aus.pdf,
Abruf: 2008.04.08

[Bor08a] Borland Software Corporation:
Borland SilkPerformer
Load and Performance Testing,
http://www.borland.com/resources/en/pdf/products/silk/silkperformer_datasheet.pdf,
Abruf: 2008.03.13

[Bor08b] Borland Software Corporation:
Borland SilkTest
Robust Functional Test Automation,
http://www.borland.com/resources/en/pdf/products/silk/silktest_datasheet.pdf
Abruf: 2008.08.26

[Cob08] Cobson, Aja:
Blackboard Academic Suite
Hardware Sizing Guide,
http://www.blackboard.com/europe/de/products/index.htm,
Abruf: 2008.08.26

[Com08] Compuware Corporation:
QALoad Product Detail,
http://www.compuware.com/products/qacenter/402_ENG_HTML.htm,

Abruf: 2008.08.26

[Dau07] Daum, Berthold:
Rich-Client-Entwicklung mit Eclipse 3.2
Anwendungen entwickeln mit der Rich Client Platform,
2., aktualisierte Auflage, dpunkt.verlag GmbH, Heidelberg, 2007

[Fli08] Fliege, Marko:
Systemgrobkonzept
Architekturkonzept,
https://wiki.alea.de/display/CON/Systemgrobkonzept
Abruf: 2008.09.01

[FT08] Förster, Patrice; Roth, Thomas:
Entscheidungstabelle Software,
https://wiki.alea.de/display/TST/Entscheidungstabelle+Software,
Abruf: 2008.08.26

[fro08] froglogic GmbH:
Squish for Java
DATA SHEET,
http://www.froglogic.com/download/datasheet_squishjava.pdf,
Abruf: 2008.08.26

[HKP08a] Hewlett Packard Company, Keanaaina Frances, Perez, Judy:
HP Loadrunner software Data sheet,
https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-126-17%5E8_4000_100__,
Abruf: 2008.08.26

[HKP08b] Hewlett Packard Company, Keanaaina, Frances, Perez, Judy:
HP QuickTest Professional software Data sheet,
https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-127-24%5E1352_4000_5__,
Abruf: 2008.08.26

[Hob08] Hobold, Michael Roberto:
Testautomatisierung RCP basierender Java Applikationen auf Solaris 10,
http://epub.wu-wien.ac.at/dyn/virlib/bakkWI/mediate/epub-wu-01_bf0.pdf?ID=epub-wu-01_bf0,

Abruf: 2008.08.26

[Imr08] Imrie, Alexandra:
Tool-Radar: GUIdancer,
in Javamagazin, März 2008, Software & Support Verlag GmbH, Frankfurt am Main, 2008,
S. 13

[Nor08] Northover, Steve:
SWT: The Standard Widget Toolkit
PART1: Implementation Strategy for Java Natives,
http://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html,
Abruf: 2008.08.26

[Ori08] Orientation in Objects GmbH:
Testen von Eclipse RCP-Anwendungen (SWT),
http://www.oio.de/eclipse-rcp-test-artikel.htm,
Abruf: 2008.08.26

[Qua08a] Quality First Software GmbH:
QF-Test
Das Java GUI Testtool,
http://www.qfs.de/de/info/Booklet_QF-Test.pdf,
Abruf: 2008.04.08

[Qua08b] Quality First Software GmbH:
QF-Test – Das Handbuch,
pdf-Datei als Anlage zu QF-Test

[Rot08a] Roth, Thomas:
Anforderungen Sizing,
https://wiki.alea.de/display/TST/Anforderungen+Sizing,
Abruf: 2008.08.26

[Rot08b] Roth, Thomas:
Testplan Sizing,
https://wiki.alea.de/display/TST/Testplan+Sizing,
Abruf: 2008.08.26

[Rot08c] Roth, Thomas:
Nutzungsdaten,

https://wiki.alea.de/display/TST/Nutzungsdaten,
Abruf: 2008.08.26

[Rub08] Rubel, Dan:
Automating GUI Testing for Eclipse RCP Applications,
http://instantiations.com/PDFs/published/gui_testing_stp.pdf,
Abruf: 2008.08.26

[SBS07] Sneed, Harry M. (MPA), Baumgartner, Manfred, Seidl, Richard:
DER SYSTEMTEST
ANFORDERUNGSBASIERTES TESTEN VON SOFTWARE-SYSTEMEN,
Carl Hanser Verlag, München Wien, 2007

[SW02] Sneed, Harry M., Winter, Mario:
Testen objektorientierter Software
Das Praxishandbuch für den Test objektorientierter Client/Server-Systeme,
Carl Hanser Verlag, München Wien, 2002

[Tha00] Thaller, Georg Erwin:
Software-Test
Verifikation und Validation,
Verlag Heinz Heise GmbH & Co KG, Hannover, 2000

[Trö08] Trölenberg-Buchholz, Helga:
ALEA Commerce Suite – Technik und Entwicklung,
https://wiki.alea.de/display/INT/Vertrieb+und+Marketing/ALEA+Technik+Juli+2008.ppt,
Abruf: 2008.09.11

[Vig05] Vigenschow, Uwe:
Objektorientiertes Testen und Testautomatisierung in der Praxis
Konzepte, Techniken und Verfahren,
dpunkt.verlag GmbH, Heidelberg, 2005

[Wil08] Wilhelm, Thomas:
Java SWT – The Standard Widget Toolkit,
http://erde.fbe.fh-weingarten.de/keller/Downloads/grabo/java/JavaSWT.pdf,
Abruf: 2008.08.12

[Wul08] Wulz, Dieter:
Anwendung unter Last,

in Javamagazin, März 2008, Software & Support Verlag GmbH, Frankfurt am Main, 2008,
S. 15 – 18

[OV08a] o.V.:
2. Performance, 2008
http://www.performance-test.de/performancetests-begriffsdefinitionen.htm,
Abruf: 2008.08.11

[OV08b] o.V.:
Rich Client Platform,
http://wiki.eclipse.org/Rich_Client_Platform,
Abruf: 2008.08.08

[OV08c] o.V.:
Performance – die Stunde der Wahrheit,
http://www.computerwoche.de/heftarchiv/2005/8/1050573/,
Abruf: 2008.08.12

Anlagenverzeichnis

Anlage A1

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine erste Projektarbeit mit dem Thema:

Testautomatisierung
für Software mit einem SWT-Rich-Client

ohne fremde Hilfe angefertigt habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und

3. dass ich meine Projektarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Gera, den 02.10.2008 _____

Ort, Datum Unterschrift

Anlage A2

ALEApedia / Entscheidungstabelle Software

Bearbeiten Hinzufügen Werkzeuge

Hinzugefügt von Thomas Roth, zuletzt bearbeitet von Patrice Förster am Sep 22, 2008 (Änderung anzeigen)

Stichwörter:

Bearbeiten

Diese Entscheidungstabelle führt die Kriterien für die untersuchte Software auf.
Grundsätzliche Kriterien sind Aufwand, Nutzen, Risiken. Weitere Kriterien werden entsprechend der Anforderungen definiert und bewertet.

Beschreibung	Automated GUI Recorder	Fit	The Grinder	Borland® SilkPerformer®	Borland® SilkTest®	GUI-Dancer	HP LoadRunner	HP QuickTest Professional 9.5	QALoad	QF-Test	Squish for Java
Firma/Person	Eclipse Community	James Shore	Paco Gómez Philip Aston	Borland	Borland	Bredex GmbH	HP	HP	Compuware Corporation	Quality First Software GmbH	froglogic GmbH
kommerziell	nein	nein	nein	ja	ja	ja	ja	ja	ja	ja	ja
Kosten/Lizenz	Eclipse Public License	General Public License	Lizenzen	nach Anfrage	Named User: 4.500 € Concurrent User: 9.000 € Runtime Concurrent: 4.000 €	Lizenztypen & Preise	5-stelliger Preis	-		Lizenztypen & Preise	Lizenztypen & Preise
SWT-RCP-Support	✓	✗	✗	✗	✓ **nur bis Version 3.2**	✓	✗	✓	✗	✓	✓
Erstellung von Makros/Scripts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Analyse und Auswertungswerkzeuge	✗	✗	✓	✓	✓	✓	✓	✗	✓ extra Suite, z.B. Vantage	✓	✗
Multi-User-Support	✗	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓
Batch-Betrieb	✗	✗	✓	✓	✓	✓ per Command Line Interface (für jeden Client muss ein CLI (GUIdancer) gestartet werden)	✓	✗	✓	✓	✓
Monitoring	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗
HTTP-Support (DWH)	✗	✗	✓	✓	✓	✓ nur mit IE	✓	✗	✓	✓ mit Edition 'QF-Test/swt+web'	✓ mit Edition 'Squish for Web'
Status	nicht geeignet	nicht geeignet	möglichweise für DWH verwendbar	nicht geeignet	**möglicher Kandidat** SWT 3.3 Support ca. in 6-7 Wochen (Ende Mai, Anfang Juni)	**möglicher Kandidat**	nicht geeignet	nicht geeignet	nicht geeignet	**möglicher Kandidat**	**möglicher Kandidat**