



Hochschule für Angewandte Wissenschaften Hamburg  
*Hamburg University of Applied Sciences*

# Bachelorarbeit

Can Eskikaya

Design und Implementierung eines POS-Systems

Verkaufssoftware zum Einsatz auf spezifischer  
Kassenhardware in einer Kennzeichenprägestelle

Can Eskikaya

Design und Implementierung eines POS-Systems

Verkaufssoftware zum Einsatz auf spezifischer  
Kassenhardware in einer Kennzeichenprägestelle

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung  
im Studiengang Technische Informatik  
am Department Informatik  
der Fakultät Technik und Informatik  
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Bettina Buth  
Zweitgutachter : Prof. Dr. Olaf Zukunft

Abgegeben am 22. September 2011

## **Can Eskikaya**

### **Thema der Bachelorarbeit**

Design und Implementierung eines POS-Systems – Verkaufssoftware zum Einsatz auf spezifischer Kassenhardware in einer Kennzeichenprägestelle

### **Stichworte**

POS-System, Kassensoftware, Agile Softwareentwicklung, MVC, Anforderungserhebung, Design, Datenbank, Java, JDBC, Validierung, Verifikation

### **Kurzzusammenfassung**

Diese Arbeit behandelt das Design und die Implementierung einer Verkaufssoftware zum Einsatz auf einem POS-System. Es werden Grundlagen zur Entwicklung aufgezeigt. Die Anforderungserhebung beschreibt die Anforderungen, die an das Gesamtsystem gestellt werden. Unter Einsatz der dargestellten Methoden und unter Berücksichtigung der Anforderungen, wird ein Design der Verkaufssoftware entwickelt. Das entwickelte Design wird in der Programmiersprache Java implementiert. Eine Verifikation der Implementierung erfolgt mit Hilfe von JUnit Tests und einem Systemtest nach Testplan.

## **Can Eskikaya**

### **Title of the paper**

Design and implementation of POS-Systems – sales software to use on specific cash hardware at an plate stamping business

### **Keywords**

POS-System, checkout counter software, agil software development, MVC, requirement elicitation, Design, Database, Java, JDBC, validation, verification

### **Abstract**

This thesis deals with the design and implementation of sales software for use on a POS system. There are fundamentals pointed to development. The requirements elicitation describes the requirements that are imposed on the overall system. Using the described methods and taking into account the requirements, a design of sale software is developed. The developed design is implemented in the Java programming language. A verification of the implementation is done with the help of JUnit tests and a system test after test plan.

# Inhaltsverzeichnis

<b>1. Einführung</b>	<b>2</b>
1.1. Zielsetzung der Arbeit . . . . .	4
1.2. Aufbau der Arbeit . . . . .	4
<b>2. Grundlagen</b>	<b>5</b>
2.1. POS-System . . . . .	5
2.2. Vorgehensmodell - Agile Softwareentwicklung . . . . .	9
2.3. Design Pattern . . . . .	11
2.4. Testen mit JUnit . . . . .	13
<b>3. Anforderungserhebung</b>	<b>14</b>
3.1. Anforderungen . . . . .	14
3.1.1. Anforderungen Hardware . . . . .	14
3.1.2. Anforderungen Software . . . . .	14
3.2. Software Requirement Specification . . . . .	18
3.2.1. Konzept und Rahmenbedingungen . . . . .	18
3.2.2. Beschreibung der Anforderungen . . . . .	19
3.3. Kosten Wartung / Erweiterungen . . . . .	23
<b>4. Design</b>	<b>24</b>
4.1. Toolselection . . . . .	24
4.2. Entity-Relationship-Modell . . . . .	26
4.3. Klassendiagramm . . . . .	27
4.4. Sequenzdiagramm mit Pseudocode . . . . .	29
4.5. Datenbankschema . . . . .	32
<b>5. Implementierung</b>	<b>34</b>
5.1. Toolselection . . . . .	34
5.2. Objektorientierte Programmierung . . . . .	37
<b>6. Validierung und Verifikation</b>	<b>40</b>
6.1. Integrationstest . . . . .	40
6.2. Systemtest . . . . .	47
<b>7. Zusammenfassung</b>	<b>49</b>
<b>Literaturverzeichnis</b>	<b>51</b>

<b>Abbildungsverzeichnis</b>	<b>54</b>
<b>A. Datenbankanbindung mittels PHP Server</b>	<b>56</b>
<b>Abkürzungsverzeichnis</b>	<b>61</b>

# 1. Einführung

Point of Service (POS) Systeme sind weltweit in den Bereichen Gastronomie, Hotel und dem Handel im Einsatz. Diese Systeme bilden die Basis für den Verkauf und die Verwaltung der Buchhaltung, da die Daten der Verkaufsvorgänge gespeichert und aufbereitet werden. Diese Systeme bestehen meist aus folgenden Kernkomponenten:

- Touchscreen (Abb. 1.1)
- Recheneinheit (PC) (Abb. 1.2)
- Linedisplay (Abb. 1.3)
- Kassenlade (Abb. 1.4)
- Barcodescanner (Abb. 1.5)
- Kartenlesegerät
- Thermodrucker (Abb. 1.6)
- POS-Software

Welche Komponenten benötigt werden, ist von den Bedürfnissen der Verkaufsstelle abhängig. In der Gastronomie findet man z. B. keinen Barcodescanner und kein Linedisplay, dafür aber meist ein mobiles Kartenlesegerät. Im Einzelhandel hingegen kommen die oben gelisteten Geräte häufig zum Einsatz. Durch die Verwendung dieser Systeme wird die Effizienz des Kassiervorganges und die der Arbeitskraft gesteigert. Zusätzlich erhöht sich die Kundenzufriedenheit, da der Kassiervorgang erheblich beschleunigt und die Wartezeit dadurch vermindert wird.

Die eingesetzte Software ist aus Kostengründen oft eine Standardimplementierung für Gastronomie, Hotel oder den Handel. Diese bietet den Vorteil, dass sie günstig und erprobt sind. Nachteile sind die beschränkten Möglichkeiten. Individuelle Wünsche werden nicht berücksichtigt, da die Software nicht für den Nutzer geändert wird. Für optimale Effizienz und umfangreiche Leistung, ist das Design und die Implementierung für das jeweilige Einsatzgebiet ratsam.

In einer Kennzeichenprägestelle werden KFZ-Kennzeichen hergestellt, dies geschieht mittels einer Presse, Klotzwerkzeugen welche die Buchstaben und Ziffern abbilden und einer Heißsiegelmaschine, die für den Farbauftrag zuständig ist. Zusätzlich werden Kennzeichenrohlinge in verschiedenen Größen benötigt. Die Klotzwerkzeuge werden in der Kombination des gewünschten Kennzeichens in eine Schablone gesetzt. Der Kennzeichenrohling wird zwischen die Klotzwerkzeuge gelegt und anschliessend unter hohem Druck

in der Presse geprägt. Nach dem Prägevorgang sind die Buchstaben und Ziffern erhaben, worauf mittels der Heißsiegelmaschine die Farbe auf das Kennzeichen aufgebracht wird. Bei diesem Verfahren gibt es keine Trocknungszeit, die Farbe ist sofort fest und kann nicht abgerieben werden. Der gesamte Herstellungsvorgang dauert ca. 3 - 5 Minuten, so dass der Verkauf unmittelbar nach dem Auftrag der Farbe stattfindet.

Außer Kennzeichen werden Kennzeichenverstärker, Funschilder und Schrauben gehandelt. Oft kommt es vor, dass Zusatzleistungen angeboten werden, wie die Annahme von Paketen, als Dienstleister eines Paketdienstes.



Abb. 1.1.: *Berührungsempfindlicher Bildschirm [26]*



Abb. 1.2.: *Recheneinheit mit Betriebssystem und POS-Software [26]*



Abb. 1.3.: *Kundenanzeige für Informationen über den Verkaufsvorgang (z.B. Preisangabe) [26]*



Abb. 1.4.: *Kassenslade zum sicheren verwahren des Geldes [26]*



Abb. 1.5.: *Handscanner zum erfassen von Strichcodes [26]*



Abb. 1.6.: *Kassendrucker zum erstellen von Kundenquittungen [26]*

## 1.1. Zielsetzung der Arbeit

Ein schneller und unkomplizierter Registriervorgang durch ein geeignetes Kassensystem kann den Verwaltungsaufwand verringern und den Verkaufsvorgang stark beschleunigen. Die Aufgaben der Kassensoftware bestehen aus der Erfassung der Kassivorgänge, die Verwaltung des Lagerbestandes und der Kundendaten, sowie die Aufbereitung der Daten für die Buchhaltung.

Diese Arbeit befasst sich mit der Architektur, dem Design und der Implementierung einer Kassensoftware zum Zwecke der Integration in das operative Geschäft einer Kennzeichenprägestelle. Ziel ist es eine optimale Lösung für den Verkauf der angebotenen Artikel, für die Erfassung der Kassivorgänge, die Verwaltung des Lagerbestandes und der Kundendaten zu finden. In dieser Arbeit wird anhand des erstellten Designs eine lauffähige Implementierung entwickelt, die in einer Kennzeichenprägestelle zum Einsatz kommen wird.

## 1.2. Aufbau der Arbeit

Die Konzepte dieser Arbeit werden in **Kapitel 2** vorgestellt. Es wird hierbei auf Grundlagen des POS-Systems, zu Vorgehensmodellen, Design Pattern und JUnit eingegangen.

In der Anforderungserhebung, die im **dritten Kapitel** vorgestellt wird, sind die Bedingungen aufgezeigt, die an das System gestellt werden. Dazu gehören auch die zu erwartenden Kosten für Wartung und Erweiterung.

In **Kapitel 4** wird das entwickelte Design mit Zuhilfenahme der UML und einem Sequenzdiagramm mit Unterstützung von Pseudocode dargestellt. Das Design beinhaltet das Funktions- und Klassendesign, sowie das Datenbankmodell.

Eine Übersicht zur Implementierung wird in **Kapitel 5** gegeben. Hier ist die Auswahl des Entwicklungswerkzeugs und das verwendete Design Pattern beschrieben.

Den Funktionstest der implementierten Komponenten und der im Anschluss gefolgte Systemtest wird in **Kapitel 6** durchgeführt.

Den Abschluss der Arbeit bildet **Kapitel 7** mit einer Zusammenfassung.



## 2. Grundlagen

Das folgende Kapitel beschreibt Grundlagen des POS-Systems, zu Vorgehensmodellen, zu Design Pattern und JUnit Testverfahren.

### 2.1. POS-System

Der Funktionsumfang und die Architektur der Point of Service Systeme sind auf das jeweilige Einsatzgebiet auszulegen. Das System wird individuell aus den benötigten Komponenten zusammengestellt. Die verschiedenen Hardwarekomponenten im POS-Systembereich werden von unterschiedlichen Herstellern entwickelt, produziert und vermarktet. Durch diese Vielfalt kam es zu einem Zusammenschluss von Hard- und Softwareherstellern, die eine gemeinsame Schnittstelle entwickelten. Diese sorgt nun für die Kompatibilität der Systeme. Die Schnittstelle UnifiedPOS ist die architektonische Spezifikation von OPOS und JavaPOS. UPOS wird entwickelt und beschrieben von der *National Retail Federation*. OPOS (OLE for POS) ist die Schnittstelle für COM fähige Programmiersprachen auf dem Betriebssystem Windows seit 1996. Die Schnittstelle JavaPOS ist für Java das, was OPOS für Windows ist und deshalb auf jedem Betriebssystem welches Java unterstützt, lauffähig ist. Die erste JavaPOS Version wurde 1996 veröffentlicht.

Die Standardisierung der Systemsoftware gewährleistet die Integration von POS Geräten unterschiedlicher Hersteller innerhalb einer Anwendungsumgebung ohne dabei die Anwendung auf die Geräte anpassen zu müssen. Somit können die POS Vertriebs Händler die für Ihren Zweck bestmögliche Hardware zusammenstellen. [12][17]

Die Einsatzgebiete der Systeme sind in drei Kategorien unterteilt:

- **Handel**  
Sämtliche Artikel, die in den Verkauf kommen, sind dem Kassensystem bekannt. Durch ein integriertes Warenwirtschaftssystem werden die Artikelbestände aktuell gehalten. Es können Statistiken über die Umsätze und verkauften Artikel erzeugt und ausgewertet werden.
- **Gastronomie**  
Die Anzahl der Tische, die Verwaltung der Kellner sowie die Bestellungen der Kunden und der automatischen Weiterleitung in die Küche stellen die Hauptanforderungen an ein Gastronomie POS System dar.

- **Hotel**

Die Verwaltung der Zimmer und deren Auslastung stehen im Vordergrund der Hotel POS Systeme. Die Bezahlung der Zimmer, der Sonderleistungen und ggf. Telefonnutzung müssen durch das POS System gewährleistet werden.

Der UnifiedPOS Standard beschreibt und beinhaltet

- **die Übersicht der UnifiedPOS Peripherie**
- **eine textuelle Beschreibung des Interfaces der Gerätefunktionen**
- **die UML Terminologie & Diagrammbeschreibung für die jeweilige Gerätekategorie**  
Die UML Terminology & Diagrammbeschreibung beschreibt die Beziehung zwischen den Klassen/Interfaces und den Objekten im System.

Der UnifiedPOS Standard beschreibt in der Version 1.13 eine Anzahl von 36 Gerätekategorien (vgl. Abb. 2.1). Im folgenden ist ein Auszug der POS-Komponenten beschrieben.

- **Touchscreen** (Abb. 1.1)  
Die Bedienung erfolgt meist ausschließlich über ein Touchscreen (MMI) Mensch Maschine Interface.
- **Thermodrucker** (Abb. 1.6)  
Ein Quittungsausdruck erfolgt für die schnelle Bearbeitung auf thermosensitiven Papier. Das Format ist klein und die Druckzeit ist kurz.
- **Kassenlade** (Abb. 1.4)  
Zur Aufbewahrung des Bargeldes für den Verkaufsvorgang kommt die Kassenlade zum Einsatz.
- **Linedisplay** (Abb. 1.3)  
Auf diesem Display, welches überwiegend aus einer Dot-Matrix Display besteht, wird der Kunde begrüßt, der zu zahlende Betrag und die Verabschiedung wird angezeigt.
- **Scanner** (Abb. 1.5)  
Zum einlesen von einem Strichcode kommen Barcodelesegeräte zum Einsatz. Das System erkennt den Artikel ohne das der Bediener diesen Artikel im System eingeben muss.
- **Kartenlesegerät**  
Zur Bezahlung mit EC oder auch Kreditkarten, werden Kartenlesegeräte genutzt. Der Bezahlvorgang wird vom System verarbeitet.

- |                      |                                  |                            |
|----------------------|----------------------------------|----------------------------|
| ■ Belt               | ■ Electronic Value Reader/Writer | ■ Pin Pad                  |
| ■ Bill Acceptor      | ■ Fiscal Printer                 | ■ Point Card Reader/Writer |
| ■ Bill Dispenser     | ■ Gate                           | ■ POS Keyboard             |
| ■ Biometrics         | ■ Hard Totals                    | ■ POS Power                |
| ■ Bump Bar           | ■ Image Scanner                  | ■ POS Printer              |
| ■ Cash Changer       | ■ Item Dispenser                 | ■ Remote Order Display     |
| ■ Cash Drawer        | ■ Keylock                        | ■ RFID Scanner             |
| ■ CAT                | ■ Lights                         | ■ Scale                    |
| ■ Check Scanner      | ■ Line Display                   | ■ Scanner                  |
| ■ Coin Acceptor      | ■ MagStripe Reader               | ■ Signature Capture        |
| ■ Coin Dispenser     | ■ MICR                           | ■ Smart Card Reader/Writer |
| ■ Electronic Journal | ■ Motion Sensor                  | ■ Tone Indicator           |

Abb. 2.1.: UnifiedPOS unterstützte Gerätekategorien [11]

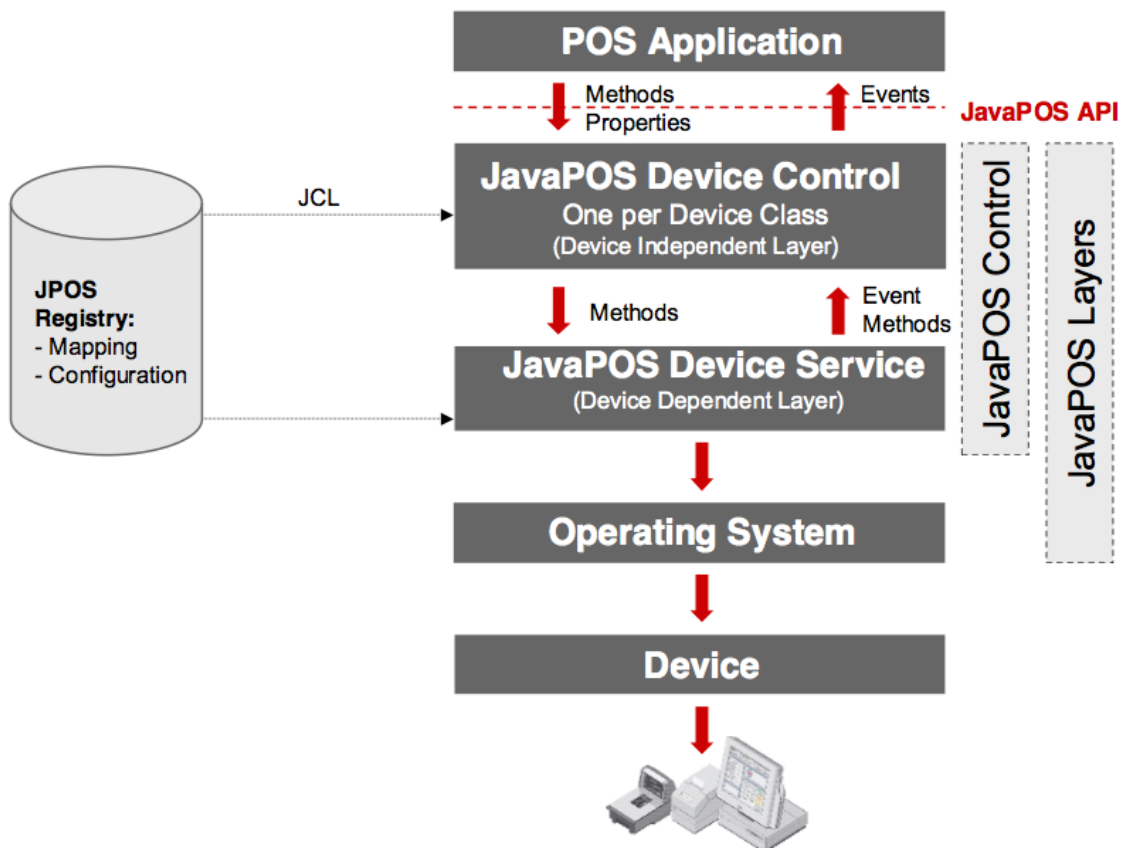


Abb. 2.2.: JavaPOS Architektur [25]

Die *JavaPOS Architektur* ermöglicht eine schnelle und einfache Implementierung der gewünschten POS Hardware (vgl. Abb. 2.2). Die *JavaPOS API* stellt alle Methoden zur Nutzung der Hardware zur Verfügung. Der Methodenaufruf wird vom *JavaPOS Device Control* verarbeitet. Der *JavaPOS Device Control* ist der geräteunabhängige Teil der *JavaPOS Architektur*. Durch die *JPOS Registry* weiß der *JavaPOS Device Control* welcher *JavaPOS Device Service* für den Methodenaufruf zuständig ist und leitet den Aufruf weiter. Der *JavaPOS Device Service* ist geräteabhängig und bekommt die Konfigurationsdaten aus der *JPOS Registry*. Der *JavaPOS Device Service* kann über das Betriebssystem mit dem angeschlossenen Gerät kommunizieren. Über Events oder Methoden, informiert der *JavaPOS Device Service* den *JavaPOS Device Control* über den Erfolg der Ausführung des Befehls oder sendet angeforderte Daten zurück. Der *JavaPOS Device Control* sendet ebenfalls angeforderte Daten bzw. die Erfolgsmeldung über Events an die *POS Applikation*. Änderungen vom Status der angeschlossenen Geräte, wie z. B. Kassenlade geöffnet oder Druckerpapier leer, werden über Eventaufrufe der *JavaPOS Control* Schicht an die *POS Applikation* gemeldet.

## 2.2. Vorgehensmodell - Agile Softwareentwicklung

Ein Vorgehensmodell ist eine abstrakte Beschreibung von Softwareprozessen, die unterteilt ist in Hauptprozess und Unterprozesse.[1]

Es gibt verschiedene Varianten von Vorgehensmodellen:

- **Arbeitsablaufmodelle**  
Abfolge von Aktivitäten (menschliche Handlungen) innerhalb des Prozesses zusammen mit Eingaben, Ausgaben und entsprechenden Abhängigkeiten.
- **Datenfluss- oder Aktivitätsmodelle**  
Menge von Tätigkeiten, die Datenumwandlungen realisieren.
- **Rolle/Aktion-Modelle**  
Stellt Rolle und Aktivitäten der Menschen dar, die in den Softwareprozess eingebunden sind.

Folgend eine Auswahl von Prozess-Modellen:

- **Wasserfallmodell**  
Gliederung des Projekts in eine Folge von Aktivitäten.
- **Ergebnisorientiertes Phasenmodell**  
Gliederung in eine Folge von Zeitabschnitten, ein Phasenabschluss durch überprüfbare Meilensteine.
- **Wachstumsmodell**  
Gliederung in eine Folge von Lieferschritten.
- **Spiral-Modell**  
Gliederung in eine zyklische, am Risiko orientierte Folge von Entwicklungsschritten.
- **Prototypen**  
Einsatz in verschiedenen Modellen möglich, hauptsächlich zur Risikoabschätzung und Risikosteuerung.
- **Agile Software-Entwicklung**  
Schlanke Software-Entwicklungs-Prozesse.

[15]

## Agile Softwareentwicklung

Die Agile Softwareentwicklung beschreibt den Einsatz von schnellen und beweglichen Prozessen in der Softwareentwicklung. Bei einer Softwareentwicklung kann dies in einigen Teilen des Projektes zum Einsatz kommen oder auf den gesamten Softwareentwicklungsprozess.

Die Zielsetzung bei Agiler Softwareentwicklung ist es, den Prozess der Entwicklung schneller, kleiner und dynamischer zu gestalten. Dies im Gegensatz zu den klassischen Vorgehensmodellen wie V-Modell oder dem Wasserfall-Modell.

Der Schwerpunkt von agilen Methoden ist Einfachheit und Geschwindigkeit. In der Entwicklung konzentriert die Entwicklungsgruppe sich auf die Funktionen, die zuerst benötigt werden. Diese werden schnell entwickelt und geliefert. Darauf folgt die Sammlung von Rückmeldungen, der Analyse der Informationen, um anschließend auf diese zu reagieren.[28]

Allen agilen Prozessen ist gemeinsam, dass sie sich zahlreicher Methoden bedienen, um die Aufwandskurve möglichst flach zu halten. Inzwischen gibt es eine Vielzahl von agilen Prozessen. Zu den Bekanntesten zählen unter anderem[22]:

- **Extreme Programming (XP)**
- **Adaptive Software Development (ASD)**
- **Cockburn's Crystal Family**
- **Dynamic System Development Method (DSDM)**
- **Feature Driven Development (FDD)**
- **Pragmatic Programming**
- **Test Driven Development (TDD)**
- **Scrum**
- **Universal Application**
- **Usability Driven Development (UDD)**

## Extreme Programming

Die Ideen zu Extreme Programming stammt von Ward Cunningham und Kent Beck, welche die ersten Pattern, die testgetriebene Entwicklung, das Responsibility-Driven Design sowie Teile der Unit-Test Frameworks XUnit entwickelten. XP besteht aus einer Reihe einfacher, aber eng zusammengehöriger Praktiken. Ziele dieser Praktiken sind hohe Produktqualität, frühzeitiges Feedback, effizienter Wissenstransfer im Team sowie Flexibilität und Änderbarkeit der erstellten Systeme. Extreme Programming hat eine strikte Vorgehensweise und definiert „Rules and Practices“ zu Planung, Design, Kodierung und Testen, auf deren Einhaltung bestanden wird. [4] [5]

## 2.3. Design Pattern

Design Pattern lösen bekannte, wiederkehrende Entwurfsprobleme. Sie fassen Design- und Architekturwissen in kompakter und wiederverwertbarer Form zusammen. Sowohl Software-Entwicklern als auch Software-Architekten bieten Entwurfsmuster wertvolle Unterstützung bei der Wiederverwendung erprobter Designentscheidungen. Sie geben Hinweise, wie vorhandene Entwürfe flexibler, verständlicher oder auch performanter gemacht werden können.

Für den Entwurf objektorientierter Systeme gelten einige Prinzipien, die auch die Basis der meisten Entwurfsmuster bilden. [3] Folgend ein Auszug der Prinzipien:

- **Einfachheit vor Allgemeinverwendbarkeit**  
Bevorzugung von einfachen gegenüber allgemein verwendbaren Lösungen.
- **Vermeidung von Wiederholungen**  
Vermeidung von Wiederholungen der Struktur und Logik, dort wo sie nicht notwendig ist.
- **Prinzip der einzelnen Verantwortlichkeit**  
Jede Klasse sollte nicht mehr als eine Aufgabe haben.
- **Offen-Geschlossen-Prinzip**  
Software-Komponenten sollten offen für Erweiterungen, aber geschlossen für Änderungen sein.
- **Prinzip der gemeinsamen Wiederverwendung**  
Die Klassen innerhalb eines Pakets sollten gemeinsam wiederverwendet werden.

## Model View Controller (MVC)

Model View Controller ist ein Architekturmuster zur Strukturierung von Software-Entwicklung in die drei Einheiten Datenmodell, Präsentation und Programmsteuerung. Ziel des Musters ist ein flexibler Programmwurf, der eine spätere Änderung oder Erweiterung erleichtert und eine Wiederverwendbarkeit der einzelnen Komponenten ermöglicht. Es ist dann zum Beispiel möglich, eine Anwendung zu schreiben, die das gleiche Modell benutzt, aber einerseits eine Windows- oder Linux-Oberfläche realisiert, andererseits aber auch eine Weboberfläche beinhaltet. Beides basiert auf dem gleichen Modell, nur Controller und View müssen dabei jeweils neu konzipiert werden.

In der Abbildung 2.3 ist das Prinzip des MVC-Patterns dargestellt. Das **Modell** hält die Daten und stellt die Logik, die den Zugriff und die Aktualisierung dieser Daten verwaltet. Die **View** stellt den Inhalt des Modells grafisch dar. Sie greift auf die Daten zu und entscheidet wie die Daten dargestellt werden sollen. Sie ist dafür verantwortlich immer aktuelle Daten anzuzeigen, auch wenn sich etwas im Modell ändert. Dies kann entweder über die Registrierung der View bei dem Modell zur Benachrichtigung von Änderungen erfolgen oder per kontinuierlicher Abfrage von Änderungen an das Modell. Der **Controller** übersetzt die Interaktionen die der Anwender in der View ausführt, in Aktionen die von dem Modell ausgeführt werden sollen. Basierend auf die Anwender-Interaktion und das Ergebniss des Modells reagiert der Controller, indem er eine entsprechende Ansicht der View auswählt.[9]

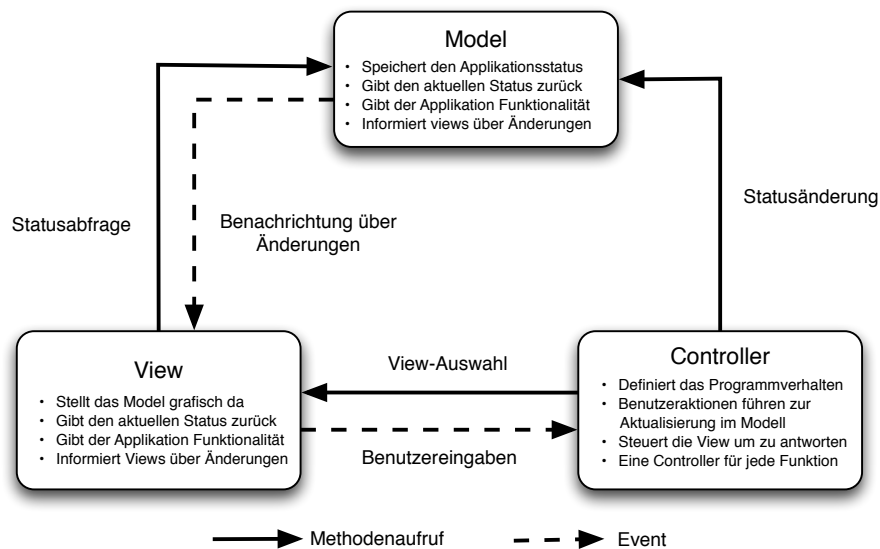


Abb. 2.3.: MVC-Pattern Konzept



## 2.4. Testen mit JUnit

JUnit ist ein Framework zum Testen von Java Klassen. Diese können automatisiert getestet werden.

Während bei Black-Box-Systemtests und Akzeptanztests die Software als Ganzes und ohne Kenntnis der Internas getestet wird, wird JUnit im Javaumfeld eingesetzt für Unit-, Komponenten- und White-Box-Tests, bei denen die Internas der zu testenden Komponente bekannt sind. Man kann JUnit aber auch als Testfirst einsetzen und die Testfälle vor der Implementierung der zu testenden Komponenten erstellen. Zu den zu testenden Komponenten werden Testklassen mit mehreren Testmethoden erzeugt. Mehrere Testklassen können zu Test-Suiten zusammengefasst werden. Mit einem Testtool können die Tests ausgeführt werden und dieses berichtet über die Ergebnisse.

JUnit ist ein frei zu verwendendes Framework und kann auf der Homepage von Junit.org heruntergeladen werden.[6]

### Funtionsweise

Ein JUnit-Test kann zwei Ergebnisse liefern; zum einen dass der Test erfolgreich abschließt, zum anderen das der Test misslungen ist. Die Ursache für das Fehlschlagen von dem Test kann durch einen Fehler *Error* oder durch ein falsches Ergebnis *Failure* verursacht werden. Beide Ursachen werden durch eine Exception signalisiert. Der Unterschied zwischen *Failure* und *Error* ist, dass ein *Failure* erwartet wird, während ein *Error* unerwartet auftritt.

### Vorgehensweise

Es gibt zwei Varianten des Vorgehens bei Junittests, zum einen kann der Programmierer zuerst die JUnit-Testfälle schreiben und dann den zu testenden Code, zum anderen kann auch erst der Programmcode implementiert und unter Kenntnis dessen die Testfälle entwickelt werden. Die Wiederholbarkeit der Testfälle ermöglicht es, die ständige Weiterentwicklung des Codes auf Fehler zu kontrollieren.

Dieses Verfahren wird auch *Testgetriebene Entwicklung* genannt und zählt zu den agilen Methoden. Die Idee dabei ist es einen fehlerarmen Code zu erzeugen, indem nichts implementiert wird, das nicht auch getestet wird.

Der Einsatz von JUnit kann die Fehlerfreiheit des erzeugten Codes nicht garantieren oder nachweisen, sondern lediglich unterstützen. Die Grenzen des Verfahrens liegen darin, dass nur solche Fehler gefunden werden können, zu deren Entdeckung die verwendeten Tests geeignet sind.[18]

# 3. Anforderungserhebung

Dieses Kapitel beinhaltet die Anforderungen an das System, die Software Requirement Specification und beschreibt die zu erwartenden Kosten für die Wartung / Erweiterung.

## 3.1. Anforderungen

Dieser Abschnitt beschreibt die Anforderungen an das System und stellt einen Technologievergleich dar. Die hier vorgestellten Anforderungen müssen für das zu entwickelnde Design berücksichtigt werden.

### 3.1.1. Anforderungen Hardware

Die Anforderungen an ein System, welches sich im Verkaufsbereich befindet, sind vielseitig. Das System befindet sich im Dauereinsatz und ist hohen äußeren Belastungen ausgesetzt, muss aber eine hohe Ausfallsicherheit gewährleisten. Des Weiteren sollten die Systeme nicht viel Platz einnehmen, da der Kassierbereichsplatz vom Verkaufs- sowie Lagerbereich abgeht. Für gute Nutzbarkeit ist ein hoher Anspruch an den berührungsempfindlichen Bildschirm gestellt. Dieser muss eine hohe Standfestigkeit haben, da er mit den Fingern betätigt wird. Die Lesbarkeit muss durch eine hohe Auflösung gewährleistet sein.

Auf dem Markt der POS-Systeme gibt es einige große und viele kleine Hersteller, die alle diesen Anforderungen mehr oder weniger genügen. [27][23]

### 3.1.2. Anforderungen Software

Hinsichtlich der Anforderungen an die Software, muss eine einfache und übersichtliche Bedienung und vor allem eine hohe Fehlertoleranz gegeben sein. Wichtige Anforderungen sind unter anderem die Wartbarkeit und der dadurch entstehende Kostenfaktor (vgl. Kapitel 3.3).

Die Programmoberfläche (GUI) sollte eine möglichst einfache Bedienung ermöglichen. Dazu ist es wichtig, dass eine strukturierte Anordnung und Übersichtlichkeit der Funktionen gegeben ist. In der Abbildung 3.1 ist zu sehen, dass alle wichtigen Informationen auf einer Maske platziert sind. Die Gruppierung der Artikel in den einzelnen Kategorien ermöglicht einen schnellen und übersichtlichen Aufruf. Die Anzeige der gewählten Artikel in einer Tabelle sorgt für eine gute Überprüfbarkeit und ermöglicht so Fehleingaben durch

versehentliches falsch wählen zu identifizieren. Der Kassiervorgang kann mit dem direkten Aufruf der Zahlart beendet werden. Des Weiteren findet man eine direkte Eingabe der Kundennummer und die Anzeige des Kunden darunter. Druckoptionen für den Quittungsdruck und Rechnungsdruck findet man ebenfalls direkt in der Hauptmaske.

Die einfache und übersichtliche Gestaltung muss sich durch das komplette Programm ziehen und in allen Masken eingehalten werden.

Die Fehlertoleranz muss über individuelle Eingabemöglichkeiten und durch das Überprüfen der Eingaben erreicht werden. Um die Fehlerquote gering zu halten ist es erforderlich beim Design darauf zu achten, dass keine unnötigen Eingaben den Benutzer verwirren bzw. in fehleranfällige Situationen herbeiführen.

Auf dem Softwaremarkt existieren viele Anbieter, die Kassensoftware für die jeweiligen Branchen anbieten. Zum Vergleich ist in der Abbildung 3.2 jeweils eine Einzelhandelsversion herangezogen worden. Die hier vorgestellten Programme sind in den aktuell verfügbaren Versionen herangezogen.

In der Abbildung 3.2 sind drei Kassenprogramme gegenübergestellt. Entsprechend dem Funktionsumfang sind die Preise der einzelnen Produkte zu vergleichen und bewegen sich auf ähnlichem Niveau. Die grafischen Oberflächen haben alle Funktionen auf einer Maske. Sie unterscheiden sich lediglich in Form und Farbe. (vgl. Abb. 3.3, 3.4 und 3.5).

Da der Funktionsumfang, der auf dem Markt befindlichen Software, sich als sehr umfangreich erweist, ist dementsprechend die GUI mit Funktionen überfrachtet. Daher liegt die Anforderung des Designs auf Schlichtheit und Funktionalität. Sehr selten genutzte oder konfiguratorische Elemente sollten nicht auf der Hauptoberfläche angezeigt und in andere Masken verschoben werden.

EURO	ZOLL/MZ	FUN	DPD
520x110	340x200	220x200	255x130
460x110	300x200	200x200	240x130
420x110	280x200	180x200	
400x110	260x200		
380x110	250x200		
370x110	240x200		
360x110	230x200		

Abb. 3.1.: Schematische Darstellung der zu erstellenden GUI

Hersteller	Name	Funktionsumfang	Preis
PosBill GmbH www.posbill.com	PosBill® Kassensoftware	Warenwirtschaft und Kassenbuch	Einzelplatz: 298,--* Netzwerk,SQL-Server Version: 598,--*
LaCash® Kassen.net	LaCash® Kassensoftware Einzelhandel	Kassenbuch	Basis 290,--* Module 190,--* (WaWi, etc.)
CSS Group Software sds-office.de	Kassensystem Einzelhandel 2008 SP3	Warenwirtschaft, Kassenbuch, Auftragsbearbeitung / Rechnungswesen	Einzelplatz: 479,00*

\*Preis +MwSt.

Abb. 3.2.: Kassensoftware auf dem Markt

Die in der Abbildung 3.2 gegenübergestellten Kassenprogramme sind in ihrer Ausstattung sehr ähnlich. *PosBill*® und *LaCash*® bieten eine Basisversion und können mit zusätzlich zu erwerbenden Modulen erweitert werden. *PosBill*® bietet eine Netzwerkversion als Komplettpaket an.

Die vorgestellten Programme bieten alle die Möglichkeit, außer der eigentlichen Kassierfunktion, die Warenwirtschaft zu erfassen.

Die Kassensysteme bieten aber keinerlei Freiraum für individuelle Anpassungen an die speziellen Anforderungen die ggf. gestellt werden.



Abb. 3.3.: GUI PosBill® [16]



Abb. 3.4.: GUI LaCash® [7]

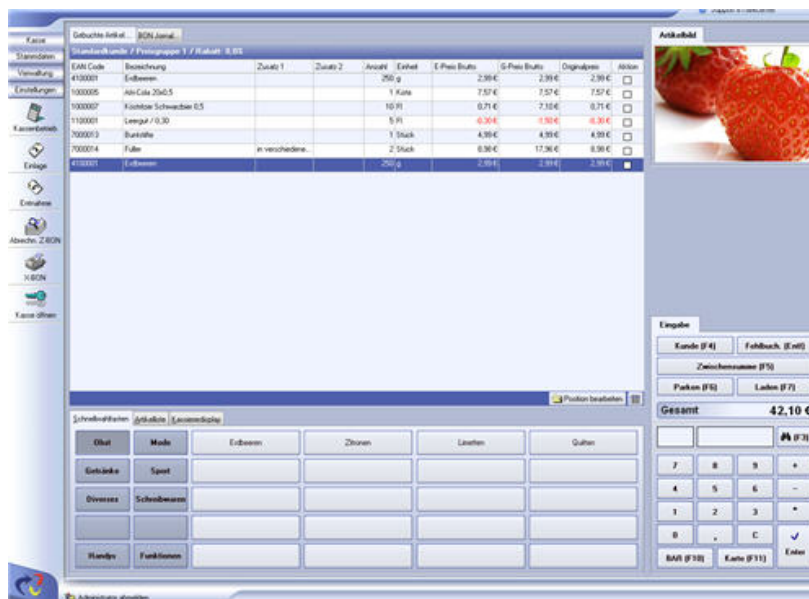


Abb. 3.5.: GUI Kassensoftware Einzelhandel 2008 [19]

Die in den Abbildungen (3.3, 3.4, 3.5) zusehenden Hauptfenster zeigen alle die gleichen Grundzüge. Es wird eine Auswahl von Buttons angezeigt, welche jeweils einen anderen Artikel repräsentieren. Durch Betätigen eines dieser Buttons, wird der entsprechende Artikel zur Artikelliste hinzugefügt. Außerdem sind sehr viele Funktionen gelistet die den eigentlichen Verkaufsvorgang nicht berühren.

## 3.2. Software Requirement Specification

Dieser Unterpunkt des Kapitels beschreibt das Konzept und die Rahmenbedingungen des Auftraggebers mit den dazugehörigen Anforderungen.

### 3.2.1. Konzept und Rahmenbedingungen

- **Ziele des Auftraggebers**

Die Software soll den Kassiervorgang in einer Kennzeichenprägestelle bearbeiten, dokumentieren und den Kassenstand sowie den Lagerbestand verwalten. Die Möglichkeit Kundendaten zu erfassen und zu speichern sollte vorhanden sein, um für Stamm- bzw. Geschäftskunden eventuelle Rabatte gewähren zu können. Ziel ist es, dass diese Vorgänge möglichst leicht und komfortabel für den Anwender zu bedienen sind, um eine hohe Effizienz des Kassiervorganges zu erreichen. Dies soll zur Folge eine höhere Rentabilität der eingesetzten Kassierkraft haben.

- **Ziele und Nutzen des Anwenders**

Die einfache und schnelle Bedienung, die dadurch erreicht wird, dass das System durch den Kassiervorgang führt, hat zum einen den Nutzen, dass der Kassiervorgang beschleunigt und zum anderen, dass die Fehlerrate reduziert wird. Da das System die verkauften Artikel dokumentiert, hat der Anwender bei der Lagerverwaltung nur noch eine Kontrollaufgabe, die Bestandsführung wird vom System übernommen. Die Tagesabrechnung des Kassenstandes wird durch das System automatisiert, wodurch auch hier mögliche Fehler minimiert werden. Ziel ist es, durch einen schnellen An- und Abmeldevorgang, die Arbeitszeit nicht für die Dokumentation und die Tagesabschlussberechnungen aufzuwenden. Dies minimiert unnötige Überstunden, des Weiteren wird die Arbeitszeit dokumentiert dies ermöglicht dem Anwender eine eigene Kontrollübersicht.

- **Systemvoraussetzungen**

Die Hardwareressourcen müssen folgende Bedingungen erfüllen bzw. nicht unterschreiten:

- **Prozessor**  
Intel Celeron M 1,5 GHz
- **Hauptspeicher**  
1 GB
- **Festplattenspeicher**  
40 GB

- **Schnittstellen**  
2 x USB2.0, 3 x EIA-232 (1 x male, 2 x female), LAN, PS/2
- **Touchscreen**
- **Thermodrucker**
- **Laserdrucker**  
USB Laserdrucker

### 3.2.2. Beschreibung der Anforderungen

Die Beschreibung der Anforderungen ist untergliedert in funktionale und nicht funktionale Anforderungen.

#### Funktionale Anforderungen

##### 1. An- und Abmeldung

Der Benutzer kann erst die Kassensoftware nutzen, nachdem er sich erfolgreich angemeldet hat.

**F-10010** Ein Benutzer kann sich anmelden durch die Eingabe einer **Zeichenkombination**.

**F-10020** Ein Benutzer kann sich jederzeit **abmelden**.

**F-10030** Ein angemeldeter Benutzer bekommt bei der **Abmeldung** einen Umsatzbericht sowie den **Kassenstand** angezeigt.

**F-10040** Ein angemeldeter Benutzer kann sich seine Anmeldezeiten **anzeigen** lassen (siehe Verwaltung F-30370).

##### 2. Verkauf

Nach erfolgreicher Anmeldung kann der Benutzer einen Kassivorgang durchführen.

**F-20050** Ein angemeldeter Benutzer kann einen Kunden anhand einer **Kundennummer** wählen.

**F-20060** Ein angemeldeter Benutzer kann einen Kunden aus einer **Kundenliste** wählen.

**F-20070** Ein angemeldeter Benutzer kann **Artikel** zur Verkaufsliste hinzufügen.

**F-20080** Ein angemeldeter Benutzer kann **Rabatte** zur Verkaufsliste hinzufügen.

**F-20090** Ein angemeldeter Benutzer kann eine **Rechnungsinformation** hinzufügen.

**F-20100** Ein angemeldeter Benutzer kann die in der Verkaufsliste stehenden Artikel einzeln **entfernen**.

**F-20110** Ein angemeldeter Benutzer kann die **Menge** des in der Verkaufsliste stehenden Artikel **verändern**.

**F-20120** Ein angemeldeter Benutzer kann, wenn die Verkaufsliste nicht leer ist, in **Bar** kassieren.

**F-20130** Ein angemeldeter Benutzer kann, wenn die Verkaufsliste nicht leer ist, per **EC** kassieren.

**F-20140** Ein angemeldeter Benutzer kann, wenn die Verkaufsliste nicht leer und der Kunde berechtigt ist, per **Überweisung/Monatsrechnung** kassieren.

**F-20150** Ein angemeldeter Benutzer kann, nach erfolgreichem Verkauf/Kassiervorgang, einen Quittierungsbeleg drucken (Thermodruck).

**F-20160** Ein angemeldeter Benutzer kann, nach erfolgreichem Verkauf/Kassiervorgang, eine Rechnung drucken (Laserdruck).

### 3. Verwaltung

Nach erfolgreicher Anmeldung kann der Benutzer Verwaltungsaufgaben erledigen. Die Möglichkeiten der Verwaltung sind von den jeweiligen Rechten des Benutzers abhängig. Benutzer, die Administratoren sind, haben erweiterte Rechte. Sie werden im folgenden mit "Benutzer\_A" gekennzeichnet.

Alle Aufgaben die ein Benutzer durchführen kann, kann ein Administrator ebenfalls durchführen. Ein Benutzer hingegen erhält nicht alle Rechte eines Administrators. **F-30170** Ein Benutzer kann sich anmelden mit Eingabe einer **Zeichenkombination**.

**F-30180** Ein Benutzer kann sich jederzeit **abmelden**.

**F-30190** Ein angemeldeter Benutzer kann aus dem Kassierumfeld zum Verwaltungsumfeld **wechseln**.

**F-30200** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann den Lagerbestand einsehen.

**F-30210** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann den Lagerbestand drucken.

**F-30220** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann verprägte Artikel melden.

**F-30230** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann einen Storno/Retoure melden.

**F-30240** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann einen tagesaktuellen Storno/Retoure Bericht einsehen.

**F-30250** Ein angemeldeter Benutzer\_A, der sich im Verwaltungsumfeld befindet kann beliebig Storno/Retoure Berichte einsehen.

**F-30260** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann Ein-/Ausgaben melden.

**F-30270** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann den



tagesaktuellen Ein-/Ausgaben Bericht einsehen.

**F-30280** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann den tagesaktuellen Kassenbericht einsehen.

**F-30290** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann den tagesaktuellen Gesamtumsatz Bericht einsehen.

**F-30300** Ein angemeldeter Benutzer\_A, der sich im Verwaltungsumfeld befindet, kann den frei wählbaren Monatsbericht einsehen.

**F-30310** Ein angemeldeter Benutzer\_A, der sich im Verwaltungsumfeld befindet, kann einen Monatsabschluss Bericht drucken.

**F-30320** Ein angemeldeter Benutzer\_A, der sich im Verwaltungsumfeld befindet, kann eine beliebige Rechnung erneut drucken.

**F-30330** Ein angemeldeter Benutzer\_A, der sich im Verwaltungsumfeld befindet, kann alle Rechnungen die noch nicht gedruckt wurden drucken.

**F-30340** Ein angemeldeter Benutzer\_A, der sich im Verwaltungsumfeld befindet, kann einen beliebige Tagesbericht erneut drucken.

**F-30350** Ein angemeldeter Benutzer\_A, der sich im Verwaltungsumfeld befindet, kann alle Tagesberichte des gewählten Monats erneut drucken.

**F-30360** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann einen beliebige Tagesbericht erneut drucken.

**F-30370** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann seinen Arbeitszeitenbericht einsehen.

**F-30380** Ein angemeldeter Benutzer, der sich im Verwaltungsumfeld befindet, kann seinen Arbeitszeitenbericht drucken.

**F-30390** Ein angemeldeter Benutzer\_A, der sich im Verwaltungsumfeld befindet, kann einen beliebigen Arbeitszeitenbericht einsehen.

**F-30400** Ein angemeldeter Benutzer\_A, der sich im Verwaltungsumfeld befindet, kann einen beliebigen Arbeitszeitenbericht drucken.

#### 4. Kundenverwaltung

Nach erfolgreicher Anmeldung kann der Benutzer Kunden verwalten.

**F-40410** Ein angemeldeter Benutzer, der sich im Kundenverwaltungsumfeld befindet, kann nach vorhandenen Kunden suchen.

**F-40420** Ein angemeldeter Benutzer, der sich im Kundenverwaltungsumfeld befindet, kann neue Kunden anlegen.

**F-40420** Ein angemeldeter Benutzer, der sich im Kundenverwaltungsumfeld befindet, kann Kunden bearbeiten.

**F-40420** Ein angemeldeter Benutzer, der sich im Kundenverwaltungsumfeld befindet, kann Kundeninformationen für den Verkauf übernehmen.

### **Nicht-funktionale Anforderungen**

#### **1. Technische Randbedingungen**

**NF-10010** Die benötigten und erzeugten Daten müssen jederzeit online und offline verfügbar sein.

**NF-10020** Die Datenübertragung zum extern gelegenen Server/Datenbank muss sicher sein (verschlüsselt).

**NF-10030** Die An- und Abmeldung eines Benutzers muss auch offline möglich sein.

**NF-10040** Alle Informationen in der Verwaltung müssen "immer" aktuell sein.

#### **2. Performance**

**NF-20010** Der Start der Software muss unter 20 Sekunden erfolgen.

**NF-20020** Der Start der Software darf nicht die Leistungskapazität des Gastsystems auslasten.

**NF-20030** Die An- und Abmeldung eines Benutzers muss einfach durchzuführen sein (max. 8 Mausklicks).

**NF-20040** Der einfache Verkaufsfall muss mit 3 Aktionen beendet sein.

**NF-20050** Die Kundensuche muss unmittelbar nach der Sucheingabe ein Ergebnis liefern.

#### **3. Ergonomie**

**NF-30010** Die angezeigten Informationen und Eingabewünsche müssen minimal gehalten werden.

**NF-30020** Die angezeigten Informationen und Eingabewünsche müssen übersichtlich gestaltet sein.

### 3.3. Kosten Wartung / Erweiterungen

Dieses Kapitel beschreibt die zu erwartenden Kosten für die laufende Wartung und ggf. der gewünschten Erweiterungen.

In der Regel wird der Wartungsaufwand mit 10%-30% der Softwarekosten berechnet. Die Software ist im Rahmen dieser Arbeit entstanden, somit liegt der Wartungsaufwand gemessen an der Zeit, bei 2,7 bis 8 Wochen im Jahr.

Der Aufwand für die Wartung der Kassensoftware ergibt sich durch das Design und die Strukturierung des Programmcodes. Der größte Aufwand kann nur durch schwerwiegende Fehler entstehen. Der normale Wartungsaufwand beschränkt sich auf das Hinzufügen von Artikeln in der Datenbank und gegebenenfalls einem *Button Element* in der der Hauptansicht. Es wäre denkbar eine Automatisierung dieser Funktion, als Erweiterung zu implementieren.

Die Möglichkeiten der Erweiterungen und dem damit verbundenen Aufwand sind den gewünschten Erweiterungen entsprechend hoch. Eine Funktion mittels der eine Kassenlade geöffnet wird und die Überwachung des Zustandes erfolgt, ist eine einfach und schnell zu realisierende Funktion, da dies über die JavaPOS Schnittstelle geschieht.

Der Aufwand für eine Erweiterung die Statistiken über Umsatz der letzten Jahre, Monate, Wochen und Tage mit Diagrammen und einer Zukunftsanalyse bereitzustellen, ist entsprechend hoch und kann mehrere Wochen oder Monate in Anspruch nehmen.

# 4. Design

Folgendes Kapitel stellt das Design des Kassensystems dar. Der erste Abschnitt beschreibt die Kriterien zur Toolselection und stellt eine Auswahl von UML-Tools gegenüber. Die weiteren Abschnitte zeigen die Beziehung der Entitäten, einen Auszug der Klassen, das Sequenzdiagramm eines einfachen Verkaufsfalles, beschrieben durch den Pseudocode und das Datenbankschema.

## 4.1. Toolselection

Die UML enthält einen Satz von Techniken zur grafischen Notation, die dabei helfen, Modelle von Softwaresystemen unter Einbeziehung ihrer Struktur und ihres Designs so zu spezifizieren, visualisieren und zu dokumentieren, dass all diesen Anforderungen gerecht wird. Auf dem Markt sind viele professionelle Tools zur Erstellung von Diagrammen verfügbar. Bei der Auswahl der Tools für die Erstellung der Diagramme waren die Bedingungen wie folgt:

- Unterstützung von UML
- Unterstützung von ER-Diagrammen
- Unterstützung von Sequenzdiagramm-Diagrammen
- Unterstützung von Klassen-Diagrammen
- Unterstützung von Export-Funktionen (PDF, JPEG,...)
- Einfache Bedienung

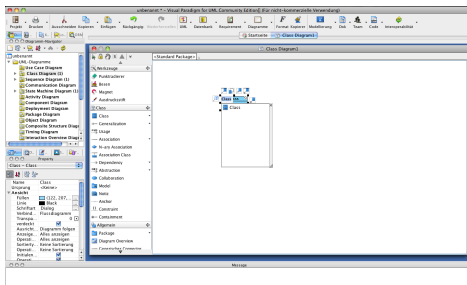


Abb. 4.1.: UI Visual Paradigm [24]

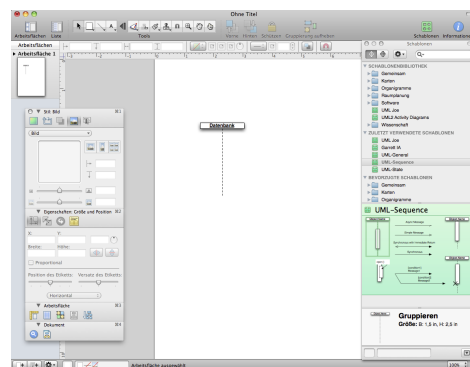


Abb. 4.2.: UI OmniGraffle 5 [20]

Tool	Hersteller	Anmerkung	Unterstützte Diagrammarten			Unterstützte Plattformen
			ER-Diagramm	Klassendiagramm	Sequenzdiagramm	
Visual Paradigm for UML	Visual Paradigm	Community Edition kostenlos	✓	✓	✓	Linux Mac OS X Windows
OmniGrafle 5	The Omni Group	Standard: \$99.99 Professional: \$199.99	✓	✓	✓	OS X 10.5 or later
EclipseUML	Omondo	UML-Plugin für Eclipse \$299; Helios/Indigo frei: Eclipse Galileo 3.5	✓	✓	✓	Java
Visio	Microsoft	Standard: 329 € Professional: 699 € Premium: 1299 €	✓	✓	✓	Windows

Abb. 4.3.: UML Tools (*Visual Paradigm* [24]; *OmniGrafle 5* [20]; *EclipseUML* [14]; *Visio* [8])

Die vier ausgewählten Designtools (vgl. Abb. 4.3) unterstützen alle Funktionen, die als Bedingung festgelegt wurden.

Die Community Edition von Visual Paradigm liefert keine Codegenerierung. Wird die Funktion gewünscht, muss eine kostenpflichtige Version des Programms erworben werden. Da dies aber nicht zu den Anforderungen für das gesuchte Designtool gehört, wurde dieses Programm näher begutachtet (vgl. Abb. 4.1). Die Software OmniGrafle 5 ist ein nicht auf UML spezialisiertes Programm, bietet aber alle Möglichkeiten diese darzustellen (vgl. Abb. 4.2). So wie Visual Paradigm for UML kann auch OmniGrafle 5 keine Codegenerierung. Die Toolselection beschränkt sich auf das Testen der beiden Programme. Visual Paradigm for UML bedarf einer längeren Einarbeitungszeit, da der Funktionsumfang sehr hoch ist. Das Programm ist nach den jeweiligen Diagrammtypen strukturiert. Es bietet viele Hilfefunktionen die das Erstellen der Diagramme beschleunigen.

Das Designtool OmniGrafle 5, bietet die Möglichkeit beliebige Template-Designstrukturen nachzuladen, liefert aber die Zeichenobjekte der UML mit. Das Programm hilft beim Platzieren und Verbinden der Grafikobjekte. Es liefert aber keine Hinweise auf die zu dem jeweiligen Objekt passenden Strukturen. OmniGrafle 5 bietet jegliche Freiheit bei der Erstellung von Grafiken, dies bedeutet aber auch, dass es keine Überprüfung auf Richtigkeit der erzeugten Diagramme bietet.

Das Designtool OmniGrafle 5, wurde zur Erstellungen der Diagramme gewählt. Dies Begründet sich aus der Freiheit, bei der Erstellungen von Diagrammen, die das Programm mit bringt. Außerdem benötigt es kaum Zeit die Funktionen, zur Erstellung von Diagrammen, zu erlernen.

## 4.2. Entity-Relationship-Modell

In der Abbildung 4.4 ist das Entity-Relationship-Modell dargestellt. Dies dient als Grundlage für das Datenbankdesign. Die Attribute der Entitäten sind aus Gründen der Übersichtlichkeit nicht auf der Abbildung zu sehen. Die Basis ist der Benutzer (Verkäufer) der einen Verkauf tätigt. Dieser Verkauf enthält Artikel, die an den Kunden zu dem Preis der in der Preisliste hinterlegt ist verkauft wird. Die Preisliste hat für den jeweiligen Kunden die Artikel. Der Kunde hat die Möglichkeit Bilder zu hinterlegen.

Des Weiteren hat der Benutzer die Arbeitszeiten und bei Fehlern in der Produktion die Verpraegt Entität.

Die Entität Kasse enthält die Summe der Bargeldverkäufe, der entsprechenden Tage die in der Verkäufe Entität hinterlegt sind.

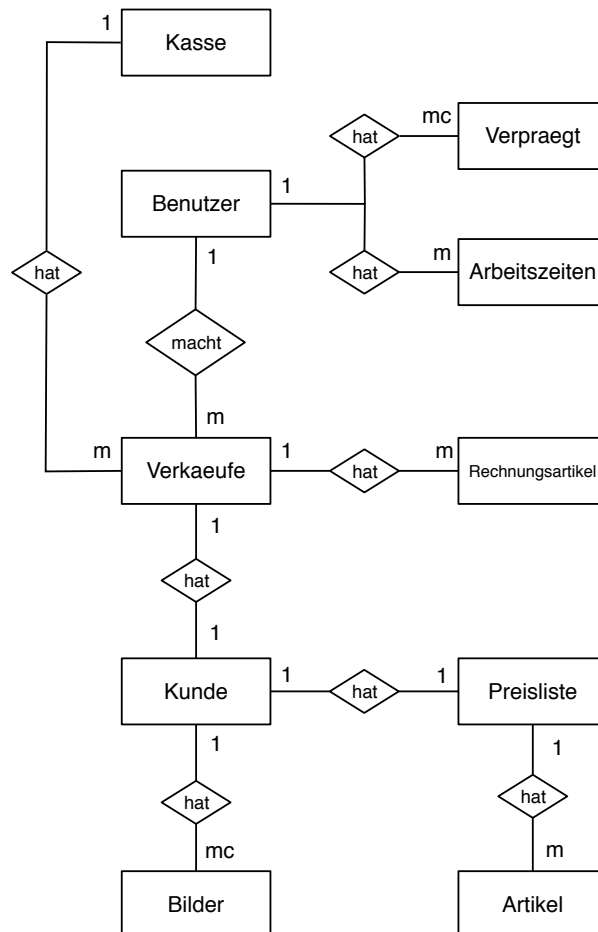


Abb. 4.4.: Entity-Relationship-Modell

### 4.3. Klassendiagramm

Der folgende Abschnitt zeigt einen Auszug des Klassendiagramms.

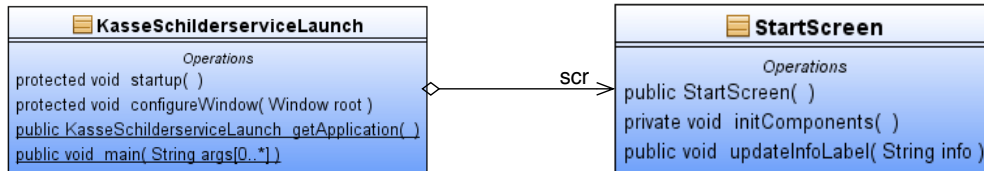


Abb. 4.5.: Auszug Klassendiagramm – Start

Die in der Abbildung 4.5 dargestellte Klasse *KasseSchilderserviceLaunch* beinhaltet die *Main()* Methode. Diese führt den Kassenstart durch und zeigt zunächst den *Startscreen* an. Ist die Kassensoftware initialisiert, wird der *Startscreen* ausgeblendet.



Abb. 4.6.: Entityklasse – Artikel

Abb. 4.7.: Druckklasse – Modul zum druck auf den Standarddrucker

Die *Artikel* Klasse, welche in der Abbildung 4.6 dargestellt ist, ist beispielhaft für die Entity-Klassen. Sie bietet nur Getter- und Setter-Methoden für ihre Attribute.

In der Abbildung 4.7 ist die *DruckRechnung* Klasse mit einer inneren Klasse *DruckenLaserTask* dargestellt, diese führt den Druckvorgang als neuen Backgroundtask aus, so dass im Kassenprogramm, nach starten des Druckauftrags, unmittelbar weiter gearbeitet werden kann.





## 4.4. Sequenzdiagramm mit Pseudocode

Das Sequenzdiagramm in der Abbildung 4.9 zeigt den Ablauf vom Programmstart und einen einfachen Verkaufsvorgang mit einer Barzahlung. Die Abläufe beim Kassieren ähneln sich stark, daher ist der hier gezeigte Vorgang exemplarisch für alle Kassiervorgänge.

Der Ablauf beim Kassenstart beginnt mit der Initialisierung. Es wird eine Verbindung zur Datenbank hergestellt. Im Anschluss folgt die PIN Abfrage zur Verifizierung des Benutzers. Mit dem eingegebenen PIN wird dann eine Abfrage an das Datenbanksystem gestellt. Das Datenbanksystem liefert einen Benutzer. Ist dieser korrekt, wird der Benutzer als angemeldet in die Datenbank geschrieben und darauf folgend Daten für den laufenden Betrieb abgefragt.

Der Benutzer bekommt das Kassiermenü angezeigt. Es wird eine Anfrage nach der nächsten Rechnungsnummer gestellt. Der Benutzer kann nun die Wahl seiner Artikel treffen und hierfür wird dann der jeweilige Preis ermittelt. Das System ist nun bereit die Aufforderung zur Zahlungsmethode entgegenzunehmen. Bei der Barzahlung wird der gegebene Betrag abgefragt und alle Informationen zum Vorgang werden in das Datenbanksystem geschrieben.

Es besteht die Möglichkeit eines Quittungsdruckes. Hier wird eine Abfrage für die benötigten Daten an das Datenbanksystem gestellt. Die empfangenen Daten führen zum Ausdruck der Quittung. Dieser Ausdruck wird protokolliert, indem der erfolgreiche Quittungsdruck zu der Rechnung in der Datenbank abgelegt wird.

Im Anschluss ist das System für einen neuen Kassiervorgang bereit.

Der folgende Pseudocode beschreibt den Ablauf des Sequenzdiagramms und soll den einfachen Verlauf eines Verkaufs erläutern.

Zunächst wird das Passwort abgefragt und damit der Benutzer identifiziert:

**Funktion:** Identifiziere Benutzer

**Zweck:** Identifizieren des Benutzers und validieren der Eingabe.

```
password = getPassword();  
if password ≠ null then  
    user = getUser(password);  
    if user ≠ null then  
        return user;  
    end if  
    return null;  
end if
```

**Ende:** Identifiziere Benutzer

Ist der Benutzer identifiziert kann der Kassiervorgang beginnen:

**Funktion:** Verkauf

**Zweck:** Der Verkauf eines Artikels an einen Kunden und kassieren des zustande gekommenen Verkaufspreises.

```
verkaufspreis;
artikelBestand = datenbank.getArtikelBestand();
kassenstand = datenbank.getKassenstand();
if user add artikel then
    artikelBestand = artikelBestand - artikel.menge();
    verkaufspreis = verkaufspreis + artikel.getPreis();
end if
if bezahlart = "Bar" and bezahlungErfolgt() = "true" then
    kassenstand = kassenstand + verkaufspreis;
end if
```

**Ende:** Verkauf

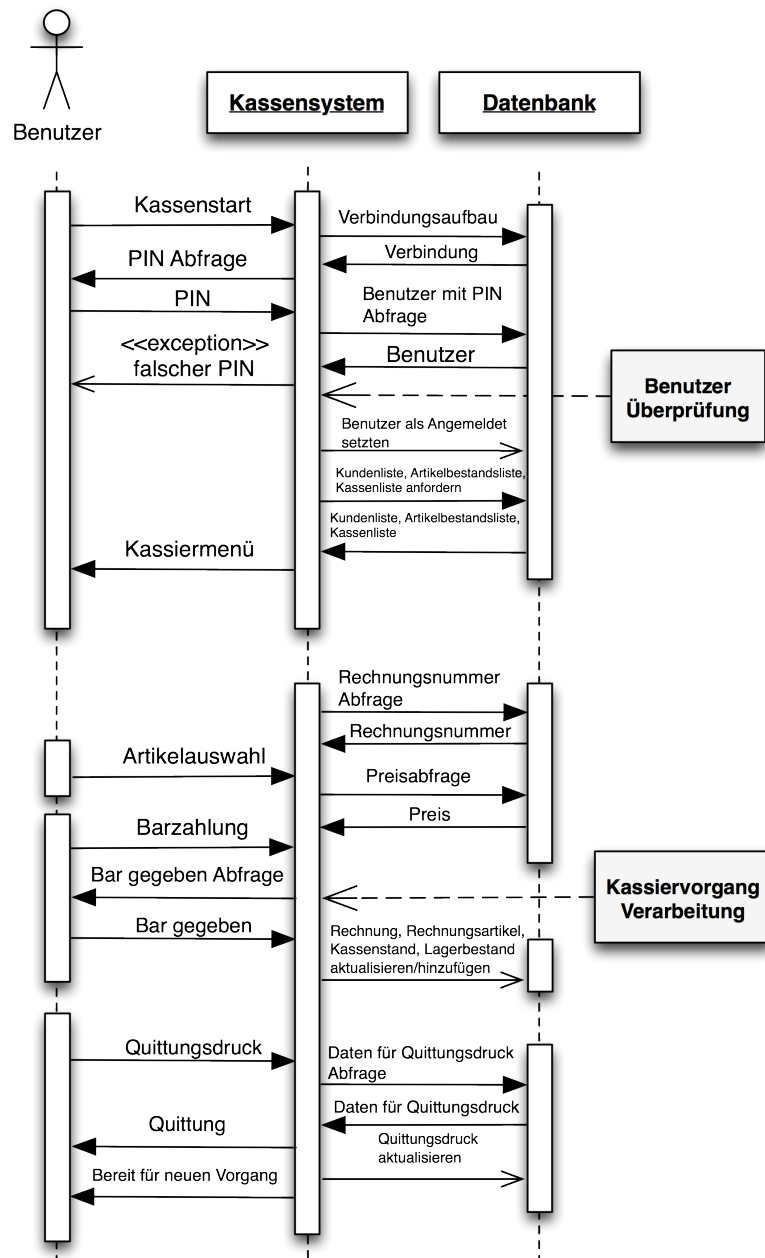


Abb. 4.9.: Sequenzdiagramm Kassenstart – Kassiervorgang

## 4.5. Datenbankschema

In der Abbildung 4.10 ist das Datenbankschema zu sehen. Es ist aus dem ER-Model entstanden und enthält alle benötigten Attribute.

Der Benutzer kann beliebig viele Verkäufe tätigen, daher ist der Benutzer mit seiner *Benutzer\_ID* in der Verkäufe Entität eindeutig identifiziert. Außerdem hat der Benutzer eventuelle Fehlprägungen und seine Arbeitszeiten, auch hier ist der Benutzer mit seiner *Benutzer\_ID* eindeutig zugeordnet.

Ein Verkaufsvorgang hat beliebig viele Rechnungsartikel und diese sind mit einer *Rechnungs\_ID* verbunden. Der jeweilige Verkaufsvorgang hat einen Kunden der über die *Kunden\_Nr* eindeutig zugeordnet wird.

Einem Kunden wird eine Preisliste zugeordnet über die *Kunden\_Nr*. Die Preisliste identifiziert die Artikel über die *Artikel\_Nr* und kann beliebig viele Artikel aufnehmen. Die Preisliste enthält zwei Schlüssel, die *Artikel\_Nr* und die *Kunden\_Nr*, diese machen die jeweiligen Preise eindeutig für den Kunden.

Die Möglichkeit des Kunden Bilder zu hinterlegen wird durch die Bilder Entität gewährleistet und die Bilder werden über die *Kunden\_Nr* zugeordnet.

Die Kasse enthält die jeweilige Tagessumme aller Barvorgänge. Sie ist mit der Verkäufe Entität über das *Datum* Attribut verbunden.

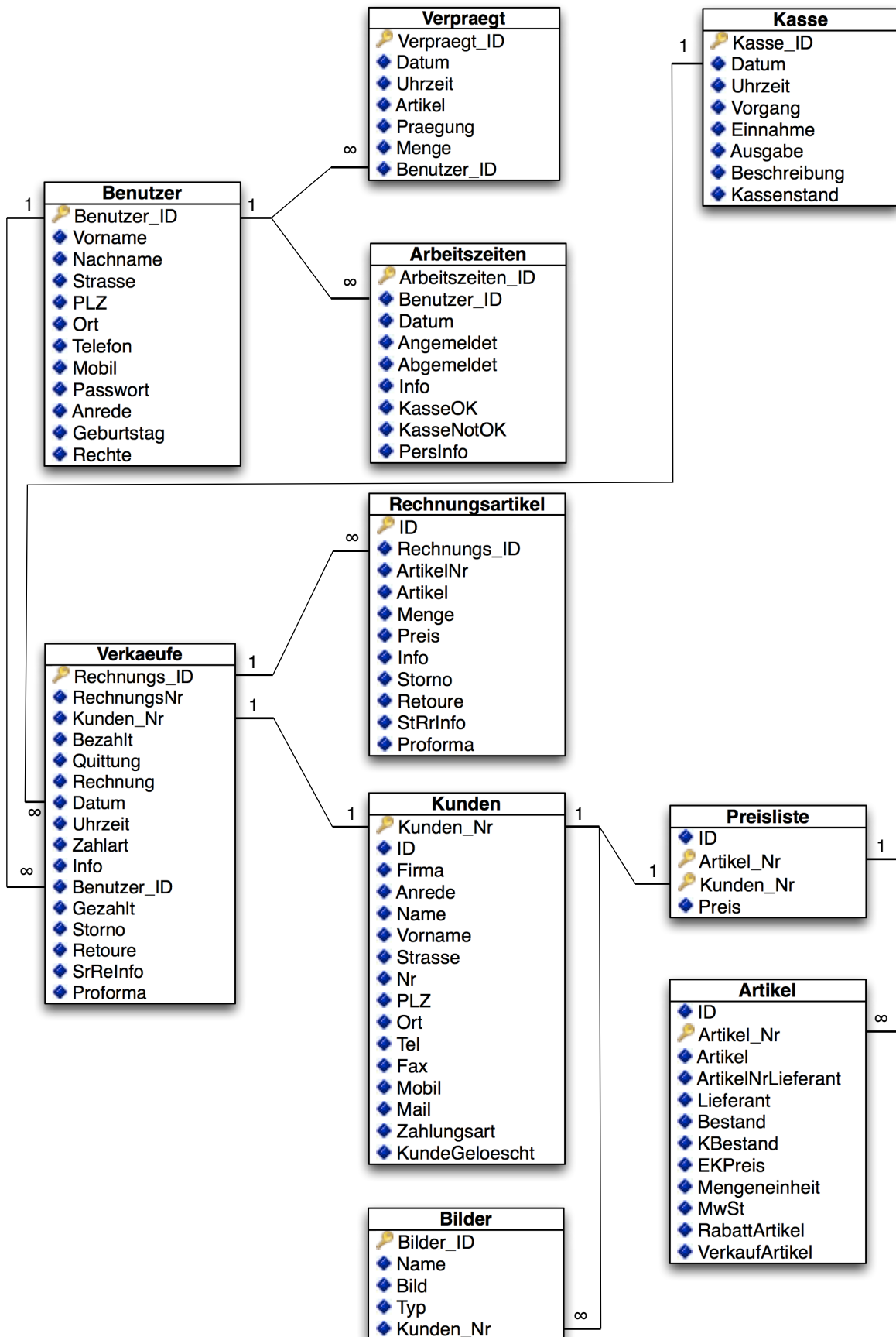


Abb. 4.10.: Datenbankmodell

# 5. Implementierung

Dieses Kapitel zeigt welche Tools bei der Programmierung zum Einsatz gekommen sind und welche Kriterien zu deren Auswahl geführt haben. Des Weiteren wird ein Teil der Implementierung mit den gewählten Tools vorgestellt.

## 5.1. Toolselection

Bei der Programmierung von Javacode gibt es verschiedene Wege lauffähige Programme zu schreiben. Ein Weg wäre über einen Texteditor den Quellcode zu entwickeln und diesen dann zu kompilieren. Ist der Funktionsumfang und der zu erwartende Quellcode nur gering, ist dies eine schnelle und unkomplizierte Möglichkeit.

Da das Kassenprogramm eine hohe Anzahl an Funktionen und Klassen beinhaltet ist die Nutzung einer Entwicklungsumgebung unerlässlich. Die zu erwartenden Fehler sind mit einer Entwicklungsumgebung schneller und unkomplizierter zu entdecken und können bei Bedarf mit einem integrierten Debugger gefunden und beseitigt werden. Während der Programmierung gibt die Entwicklungsumgebung Hinweise auf mögliche Fehler und stellt diese grafisch dar.



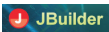





	Entwicklungs- umgebung	Lizenz	Kosten	Besonderheit
	Eclipse JDT	Eclipse Public License	kostenlos	
	NetBeans IDE	CDDL, GPL2	kostenlos	
	JBuilder	Nicht frei (proprietär)	1783,- € Enterprise 593,- € Professional	
	IntelliJ IDEA	ALv2, nicht frei (proprietär)	504,- € Commercial 210,- € Personal 84,- € Academic	Classroom, OS Projekt kostenlos Open Source kostenlos
	JDeveloper	OTN JDeveloper License, nicht frei (proprietär)	kostenlos	
	Rational Application Developer	Nicht frei (proprietär)	von 1860,- € bis 15101,- €	
	Xcode	Nicht frei (proprietär)	3,99 €	nur Mac OS
	BlueJ	GPL2, GNU mit ausnahmen	kostenlos	zum lernen/lehren

Abb. 5.1.: Auswahl an Entwicklungsumgebungen

Es gibt verschiedene Entwicklungsumgebungen, einige sind kostenlos andere müssen käuflich erworben werden. Des Weiteren unterscheiden sie sich im Funktionsumfang. Hier wird

bewusst nur auf die kostenlosen Programme eingegangen, da diese vom Leistungsumfang gleichberechtigt sind. Der Support ist bei den käuflich zu erwerbenden Entwicklungsumgebungen teilweise inklusive, bei den frei erhältlichen Entwicklungsumgebungen existieren große Gemeinschaften, die Unterstützung bei Problemen liefern.

Eine Auswahl häufig verwendeter Entwicklungsumgebungen sind in der Abbildung 5.1 zu sehen. Hier wird auf die Vorstellung aller Entwicklungsumgebungen verzichtet und nur auf die kostenlos und wohl am stärksten verbreiteten IDEs näher eingegangen. Dazu zählen Eclipse JDT und Netbeans IDE.

## Eclipse JDT

Eclipse JDT[2] ist eine quelloffene Entwicklungsumgebung, die zum programmieren verschiedener Sprachen genutzt werden kann. Das besondere ist die Erweiterbarkeit von Eclipse JDT. Es existieren eine hohe Anzahl von frei erhältlichen, aber auch kommerziellen Erweiterungen.

Eclipse JDT wird mit folgenden Komponenten für die Java Entwicklung ausgeliefert:

- Texteditor
- Compiler
- Linker
- Debugger

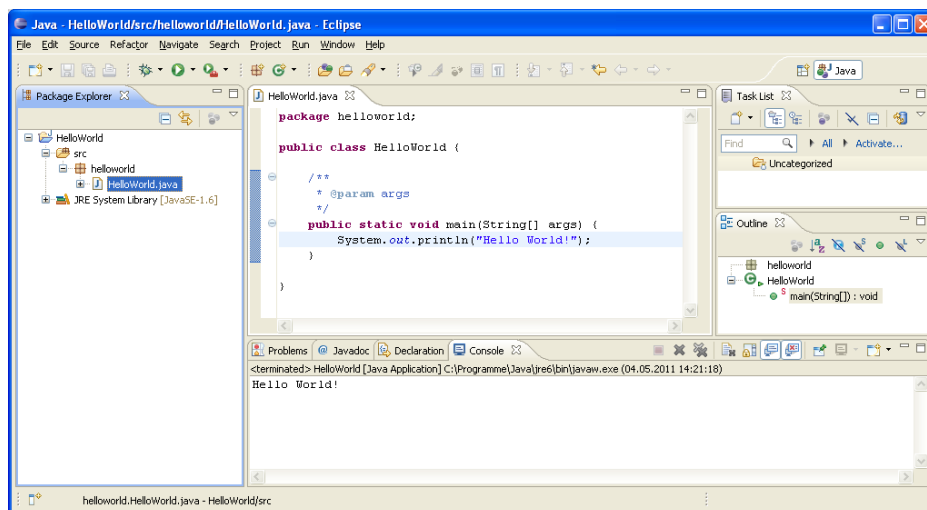


Abb. 5.2.: Eclipse JDT User Interface

Die Benutzeroberfläche ist aufgeräumt und ermöglicht eine gute Übersicht, über das aktuelle Projekt (vgl. Abb. 5.2). Die Funktionsleiste kann individuell mit weiteren Funktionen bestückt und deren Position frei gewählt werden.

Durch die große Erweiterbarkeit, kann Eclipse JDT für nahezu jedes Projekt eingesetzt werden.

## NetBeans IDE

NetBeans IDE[13] ist ebenfalls eine quelloffene Entwicklungsumgebung, die komplett in Java entwickelt wurde. NetBeans IDE kann in Packs herunter geladen werden, die für das jeweilige Vorhaben ausgerüstet sind. Auch NetBeans IDE ist erweiterbar, es existieren aber weit aus weniger Erweiterungen als für Eclipse JDT. Die Entwicklungsumgebung wird mit allen Komponenten, die zur Erstellung eines lauffähigen Javacodes benötigt wird, ausgeliefert. Eine Besonderheit ist der integrierte GUI-Builder, der es ermöglicht per Drag&Drop Benutzeroberflächen zu generieren.

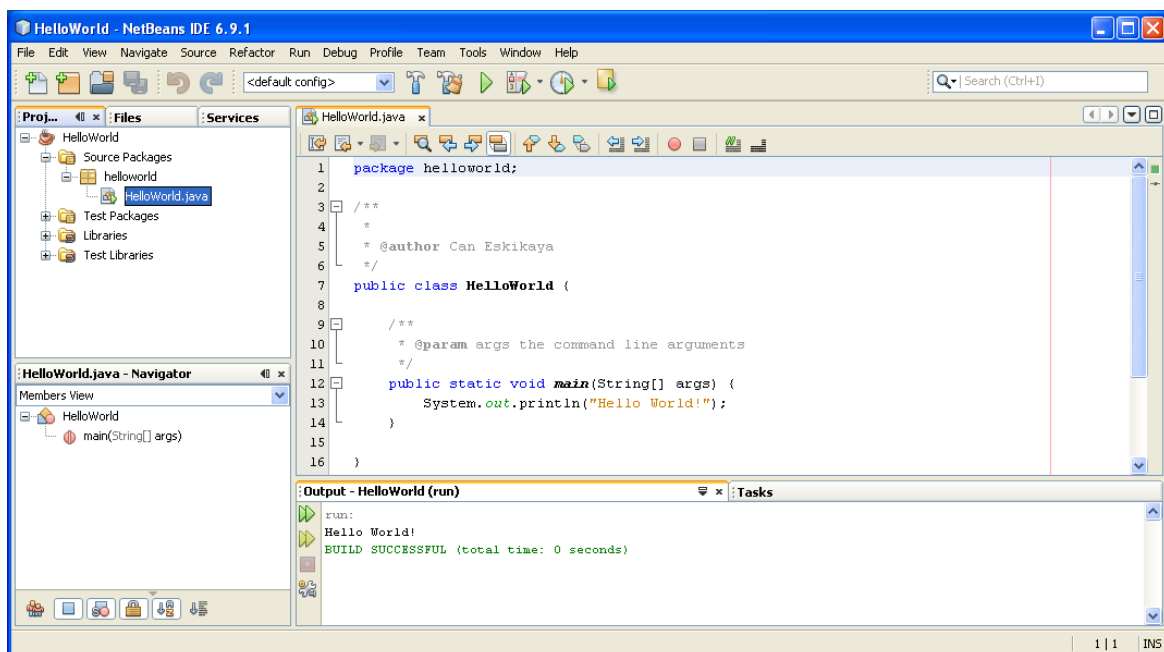


Abb. 5.3.: Netbeans IDE User Interface

Eine aufgeräumte Benutzeroberfläche ist auch bei NetBeans IDE zu finden und ermöglicht eine gute Übersicht über das aktuelle Projekt (vgl. Abb. 5.3). Die Funktionsleiste kann ebenfalls individuell mit weiteren Funktionen bestückt und deren Position frei gewählt werden.

Bei der Wahl der zu verwendenden Entwicklungsumgebung kam es hauptsächlich auf die einfache Installation und den schnellen Zugang zum Entwicklungstool an. Den Ausschlag für die Entscheidung hat der integrierte GUI-Builder gegeben. Die Wahl der eingesetzten Entwicklungsumgebung viel auf NetBeans IDE. Diese Entwicklungsumgebung bringt alle Funktionen in einem Paket mit, die für eine Java Entwicklung benötigt wird. Es ist kein Aufwand zum auffinden und konfigurieren benötigter Erweiterungen aufzubringen.



## 5.2. Objektorientierte Programmierung

Die objektorientierte Programmierung ermöglicht es gut strukturierte Programme bzw. Programmcodes zu entwickeln. In der Abbildung 5.4 ist zu sehen wie die Strukturierung des Kassensystems erstellt wurde.

Die Packagestruktur ist in einem gängigen Format angelegt. Für die eindeutige Identifizierung wurde die Domain in umgekehrter Reihenfolge für die Packagerootstruktur gewählt und anschließend der Programmname. Die einzelnen Unterordner haben eine einfache Sortierung nach Typ oder Zusammenhang.

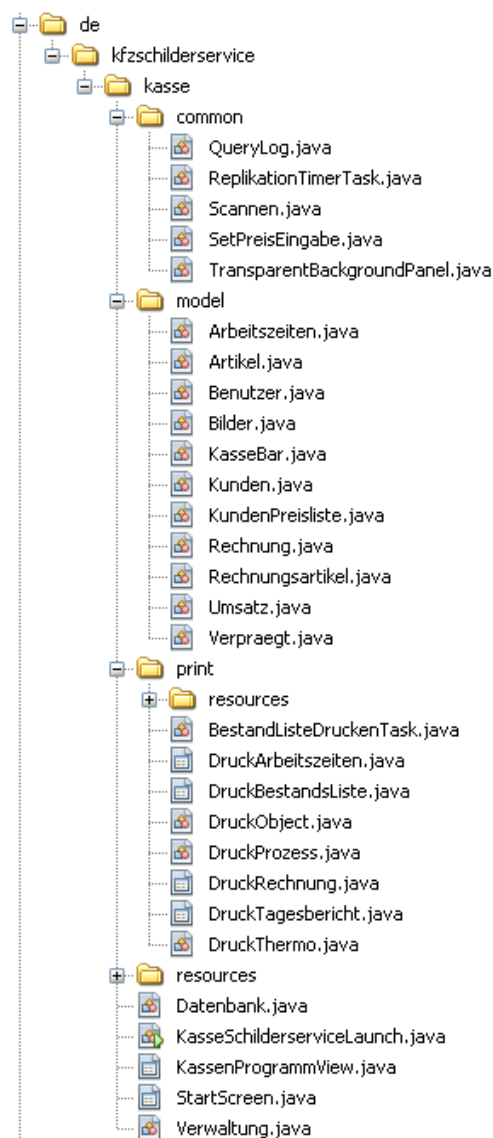


Abb. 5.4.: Package Struktur

Im folgenden sind die Teile der Implementierung beschrieben beginnend mit der GUI-Erstellung. Die Implementierung der Kassensoftware ist an das Model-View-Controller Pattern angelehnt, wobei das View mit dem Controller fast vollständig verschmolzen ist.

## View - Controller

Zunächst wird eine JFrame Klasse erstellt, diese ermöglicht es in der IDE auf die Design-Ansicht umzustellen. Hieraus hat man die Möglichkeit GUI-Elemente per Drag-and-Drop an die gewünschte Position in den Main-Frame zu setzen. Der erzeugte Code ist dann auch sofort testbar, mittels der Run Funktion, ist es möglich den Main-Frame mit dem gesetzten GUI-Element auszuführen und zur Anzeige zu bringen. Nun sind diese Elemente noch ohne Funktion. Als Beispiel sei ein Button genommen, hier erwartet man eine Aufgabe, die nach Betätigung erfüllt wird. Die Möglichkeit dies zu integrieren stellt der GUI-Builder zusätzlich bereit. Es sind unterschiedliche Möglichkeiten vorhanden, zum einen direkt auf die verschiedenen MouseEvents und oder KeyEvents zu reagieren (vgl. Abb. 5.5). Zum anderen wird die Möglichkeit geboten ein ActionEvent zu generieren, welches auch als Background-Task konfiguriert werden kann (vgl. Abb. 5.6). Es werden Methodenrumpfe generiert in denen die gewünschte Aktion dann kodiert werden kann.

```
private void addToArtikelList(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}
```

Da der eigentliche Viewcode fast komplett generiert wird, ist die Controller-Logik in der View Compilation Unit integriert. Somit ist die Logik mit dem View fest verschmolzen.

## Model

Das Model besteht aus der *Datenbank* Klasse und den Entityklassen. Die komplette Kommunikation von und zur Datenbank wird von der *Datenbank* Klasse geregelt. Werden einzelne Entitäten benötigt, wie der Preis eines Artikels, wird dieser aus der Datenbank bezogen und zurück geliefert. Werden mehrere Einträge aus der Datenbank benötigt, wie z. B. bei der Kundenkartei, wird eine Liste von Kunden generiert und diese zurückgegeben. Dieses Verfahren garantiert einen immer aktuellen Datenstand.

Die *Datenbank* Klasse baut eine Verbindung mittelst JDBC, MySQL Datenbanktreiber[10], zur Datenbank auf. Nach dem die Verbindung zur Datenbank steht, können Daten geholt, hinzugefügt und aktualisiert werden. Dies geschieht über getter, insert sowie update Methoden. Diese erzeugen die entsprechenden SQL Aufrufe und führen sie über die Datenbankverbindung aus. Die Datenbank generiert die angeforderten Daten und stellt diese als Resultset zur Verfügung. Durch das Resultset kann mittels *hasNext()* iteriert werden. Existiert kein

nächstes Element, wird *NULL* zurückgegeben. Tabellen haben mehrere Spalten und diese können aus dem jeweiligen Element aus dem Resultset abgefragt werden. Dies geschieht durch einen Aufruf ähnlich diesem: `resultsetElement.getString("COLUMN NAME")`. Es ist entscheidend, welcher Typ gefordert ist und dieser kann mit der entsprechenden getter Methode bezogen werden. Es wird zusätzlich der Name der Tabellenspalte benötigt.[21]  
Bei insert und update Aufrufen, besteht der Resultset aus einem integer Element der angibt, wie viele Spalten eingefügt bzw. aktualisiert wurden.

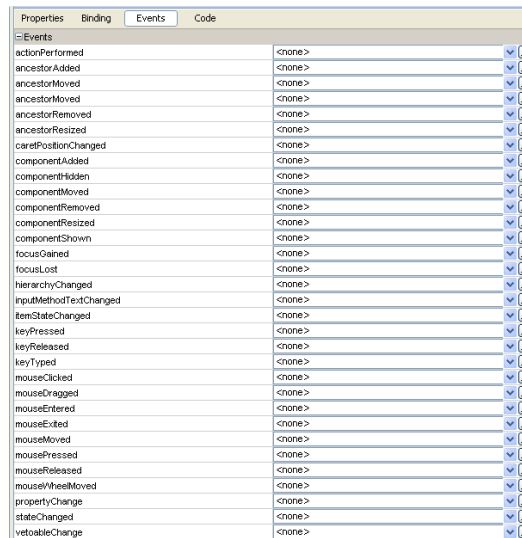


Abb. 5.5.: Liste der generierbaren Button Events

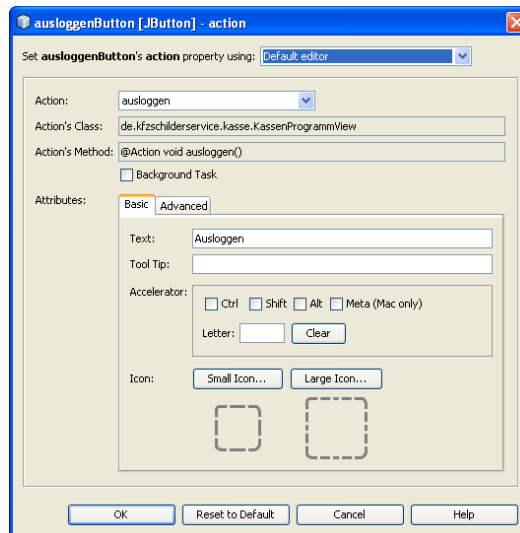


Abb. 5.6.: Action Task Menü

# 6. Validierung und Verifikation

In diesem Kapitel wird das Vorgehen der Softwaretests erläutert. Im ersten Abschnitt wird der Integrationstest der einzelnen Komponenten dargestellt. Der zweite Abschnitt zeigt den Systemtest auf, hier wird das gesamte System gegen die gesamten Anforderungen (funktionale und nicht funktionale Anforderungen) getestet.

## 6.1. Integrationstest

Bei dem Integrationstest werden eine Reihe von Einzeltest, die aufeinander abgestimmt sind, durchgeführt. Dies dient dazu, dass die von einander abhängigen Komponenten im Zusammenspiel getestet werden. Laufen diese Komponententests fehlerfrei, gelten diese als Modultest bestanden und sind für sich fehlerfrei funktionsfähig.

Die Entwicklung der in dieser Arbeit vorgestellten Software, wurde von einem Programmierer durchgeführt. Aus diesem Grund wurden laufend Integrationstest der Komponenten durchgeführt. Dies führt dazu, dass schematische Tests nur für die komplexen Komponenten erstellt wurden.

Folgende Integrationsschritte wurden durchgeführt und werden anschließend vorgestellt.

### Integrations Testschritte:

1. Testplan erstellen
2. Testfälle und Testdaten erstellen
3. JUnit Testfälle kodieren
4. JUnit Testfälle ausführen
5. Wenn Fehler im Programmcode vorhanden sind, beseitigen und erneut testen
6. Wiederholung der Testzyklen bis die Komponente erfolgreich abschließt

### Testplan

Die wichtigste zu testende Komponente ist die *Datenbank* compilation unit. Hier wird die komplette Kommunikation von und zur Datenbank abgearbeitet. Alle Methoden sind auf die richtige Funktionsweise zu testen. Die fehlerfreie Funktion dieser Schnittstelle ist für das gesamte Softwareprodukt von äusserster Wichtigkeit. Sollte ein Fehler in der Darstellung sein, so ist dies für das Gesamtsystem nicht von Relevanz. Sobald in der Datenverarbeitung oder der Datenvorhaltung ein Fehler passiert, führt dies zu einem nicht produktiv einsetzbaren System.

Die Testfälle müssen alle Methoden des *Datenbank* Moduls auf konsistente und richtige

Ausgaben testen. Die zu erwartenden Fehler bei falsch Eingaben müssen von dem *Datenbank* Modul abgefangen oder unter Warnung an die aufrufende Komponente zurück gegeben werden.

Laufen alle Test erfolgreich ist dieses Modul in der Version als fehlerfrei zu behandeln. Treten Fehler während des Tests auf, sind diese zu beheben oder bei gewollten Fehlern ist vor diesen zu warnen.

## Testfälle und Testdaten

### 1. Öffnen der Datenbankverbindung

#### Testdaten1:

```
url: `jdbc:mysql://localhost:3307/schilderservice` ;  
username: `su`; password: ``
```

#### Testdaten2:

```
url: `jdbc:mysql://localhost:3307/schilders234` ; username:  
`sdsu` ; password: `sa` ;
```

#### Testfälle:

Beim erzeugen der Datenbank mit den *Testdaten1* ist eine erfolgreiche Verbindungsherstellung zu erwarten. Der Rückgabewert ist eine *Connection*.

Beim erzeugen der Datenbank mit den *Testdaten2* ist mit einer *Connection* = NULL als Ergebnis zu rechnen.

### 2. Schließen der Datenbankverbindung

Testdaten1: Eine bestehende Datenbankverbindung

Testdaten2: Keine bestehende Datenbankverbindung

#### Testfälle:

Beim schließen der Datenbank mit den *Testdaten1* ist ein *true* als Rückgabewert zu erwarten.

Beim schließen der Datenbank mit den *Testdaten2* ist ein *true* als Rückgabewert zu erwarten, da die Datenbank bereits geschlossen ist.

Bei einem Fehlverhalten der Datenbank mit den *Testdaten1* ist ein *false* als Rückgabewert zu erwarten. Dies ist mit JUnit Tests nur schwer zu testen. Dies muss gesondert getestet werden.

### 3. Alle getter Methoden mit Parameter

Testdaten1: Parameter null oder ungültige Werte

Testdaten2: Korrekte Parameter

#### Testfälle:

Beim Aufruf der getter Methoden mit *Testdaten1* ist ein Rückgabewert mit null zu erwarten.

Beim Aufruf der getter Methoden mit *Testdaten2* ist ein Rückgabewert mit korrekten

Daten zu erwarten.

#### 4. Alle getter Methoden ohne Parameter

Testdaten1: Bestehende Datenbankverbindung

Testdaten2: Keine bestehende Datenbankverbindung

Testfälle:

Aufruf der getter Methoden mit *Testdaten1*. Der erwartete Rückgabewert muss geliefert werden.

Aufruf der getter Methoden mit *Testdaten2*. Ein Verbindungsversuch sollte ausgeführt werden und bei Erfolg den erwarteten Rückgabewert liefern. Bei Misserfolg wird ein leerer Rückgabewert bzw. null erwartet.

#### 5. Alle insert Methoden

Testdaten1: Parameter null oder ungültige Werte

Testdaten2: Korrekte Parameter

Testfälle:

Beim Aufruf der insert Methoden mit *Testdaten1* wird eine *IllegalArgumentException* erwartet.

Beim Aufruf der insert Methoden mit *Testdaten2* wird kein Rückgabewert erwartet, da die insert Methoden vom Typ *void* sind. Erwartet wird dann, dass die Daten erfolgreich in der Datenbank eingetragen sind. Ein weiterer Aufruf der entsprechenden getter Methode soll den Erfolg des Insertaufrufs überprüfen.

#### 6. Alle update Methoden

Testdaten1: Parameter ungültige Werte

Testdaten2: Korrekte Parameter

Testfälle:

Beim Aufruf der update Methoden mit *Testdaten1* wird eine *IllegalArgumentException* erwartet.

Beim Aufruf der update Methoden mit *Testdaten2* wird kein Rückgabewert erwartet. Der Erfolg wird mit der entsprechenden getter Methode getestet.

## JUnit Testfälle

Folgend sind die kodierten Testfälle für den *Datenbank* Verbindungsaufbau, Verbindungsabbau und jeweils ein kodierter Testfall beispielhaft für getter, insert und update Methoden dargestellt.

```
/**
 * Test of openDB method, of class Datenbank.
 */
@Test
public Connection testOpenDB(){
    System.out.println("conCloseLocal");
    Connection instance = Datenbank.openDB(dbAdresse, dbBenutzer,
        dbPasswort);
    assertNotNull(instance);
    instance.close();
}

/**
 * Test of conCloseLocal method, of class Datenbank.
 */
@Test
public void testConCloseLocal(){
    System.out.println("conCloseLocal");
    Datenbank instance = new Datenbank(dbAdresse, dbBenutzer,
        dbPasswort, dbAdresseExt, dbBenutzerExt, dbPasswortExt);
    boolean expResult = true;
    boolean result = instance.conCloseLocal();
    assertEquals(expResult, result);
    result = instance.conCloseLocal();
    assertEquals(expResult, result);
    instance.conCloseExtern();
}

/**
 * Test of getUser method, of class Datenbank.
 */
@Test
public void testGetUser(){
    System.out.println("getUser");
    String passwordWrong = "1234";
    String passwordCorrect = "#1199*";
```

```
Datenbank instance = new Datenbank(dbAdresse, dbBenutzer,
    dbPasswort, dbAdresseExt, dbBenutzerExt, dbPasswortExt);
Benutzer expResult = new Benutzer();
expResult.setPassword("passwordFalse");
Benutzer result = instance.getUser(passwordWrong);
Benutzer resultCorrect = instance.getUser(passwordCorrect);
assertEquals(expResult instanceof Benutzer,
    result instanceof Benutzer);
assertEquals(expResult.getPassword(), result.getPassword());
assertNotNull(result);
assertNotNull(resultCorrect);
assertNotSame(expResult.getPassword(),
    resultCorrect.getPassword());
instance.conCloseExtern();
instance.conCloseLocal();
}

/**
 * Test of getArtikelList method, of class Datenbank.
 */
@Test
public void testGetArtikelList() {
    System.out.println("getArtikelList");
    Datenbank instance = new Datenbank(dbAdresse, dbBenutzer,
        dbPasswort, dbAdresseExt, dbBenutzerExt, dbPasswortExt);
    List unexpected = new ArrayList<Artikel>();
    List result = instance.getArtikelList();
    assertNotNull(result);
    assertNotSame(0, result.size());
    assertNotSame(unexpected, result);
}

/**
 * Test of insertImage method, of class Datenbank.
 */
@Test
```

**(expected= IllegalArgumentException.class)**

```
public void testInsertImage() {
    System.out.println("insertImage");
    byte[] image = null;
```



```
String kundenNr = "";
String bildName = "";
Datenbank instance = new Datenbank(dbAdresse, dbBenutzer,
    dbPasswort, dbAdresseExt, dbBenutzerExt, dbPasswortExt);
instance.insertImage(image, kundenNr, bildName);
}

/**
 * Test of insertImage method, of class Datenbank.
 */
@Test
public void testInsertImage() {
    System.out.println("insertImage");
    byte[] image = String.valueOf(100).getBytes();
    String kundenNr = "1";
    String bildName = "Bild100";
    Datenbank instance = new Datenbank(dbAdresse, dbBenutzer,
        dbPasswort, dbAdresseExt, dbBenutzerExt, dbPasswortExt);
    instance.insertImage(image, kundenNr, bildName);
}

/**
 * Test of updateQuittungskopien method, of class Datenbank.
 */
@Test
(expected= IllegalArgumentException.class)

public void testUpdateQuittungskopien() {
    System.out.println("updateQuittungskopien");
    int id = -1;
    Datenbank instance = new Datenbank(dbAdresse, dbBenutzer,
        dbPasswort, dbAdresseExt, dbBenutzerExt, dbPasswortExt);
    instance.updateQuittungskopien(id);
}

/**
 * Test of updateQuittungskopien method, of class Datenbank.
 */
@Test
public void testUpdateQuittungskopien() {
    System.out.println("updateQuittungskopien");
```

```
int id = 5;
Datenbank instance = new Datenbank(dbAdresse, dbBenutzer,
    dbPasswort, dbAdresseExt, dbBenutzerExt, dbPasswortExt);
instance.updateQuittungskopien(id);
}
```

### **JUnit Testergebnisse zusammengefasst**

Das Ausführen der JUnit Testmethoden, führte wie *erwartet* zu Fehlern. Die anschließende Fehlerbeseitigung und Durchführung der Testdurchläufe nach Testplan erzielte den gewünschten Erfolg.

Es ist möglich, dass ein Testfall nicht entwickelt wurde der unter Umständen eintreten kann. Daher geben JUnit Tests stets keine Garantie auf Vollständigkeit. Sie bieten aber die Möglichkeit, während der Entwicklung den Entwickler auf Probleme hinzuweisen, welche dann umgehend beseitigt werden können.

## 6.2. Systemtest

Der Systemtest sieht vor, dass alle Komponenten zusammen getestet werden. Dieser Test wird auf der Kassenhardware durchgeführt. Hierzu müssen zunächst alle Komponenten die gebraucht werden installiert, konfiguriert bzw. angeschlossen sein.

Die Kassenhardware besteht aus folgenden Komponenten:

- **Rechner**  
Wincor Nixdorf BEETLE /M-II
- **Touchscreen**  
Wincor Nixdorf BA-72 (12")
- **Thermodrucker**  
Epson TM-T88III USB
- **Laserdrucker**  
USB Multifunktions Laserdrucker
- **Tastatur**  
Standard Tastatur (Layout: deutsch)

Die Kassensoftware benötigt zum Betrieb mit dem Thermodrucker eine Softwareschnittstelle, die als Softwarepaket von der Firma Epson zur Verfügung gestellt wird. Dies ist die JPOS Schnittstelle, die in dem Grundlagenkapitel vorgestellt wurde. Diese muss installiert und für den entsprechenden Thermodrucker konfiguriert werden. Ist die Konfiguration korrekt, kann der Thermodrucker mit den JPOS Java Libraries angesprochen werden.

Der Laserdrucker muss auf dem System mit den mitgelieferten Treibern installiert bzw. konfiguriert werden. Der Drucker wird von der Kassensoftware über die Java Print Service API (JPS) angesprochen und genutzt.

Der Systemtest beginnt damit, die Peripheriegeräte auf Funktionstüchtigkeit zu testen. Dies geschieht über die vom Betriebssystem bereitgestellten Mittel. Da das Kassensystem auf einem Microsoft Windows XP Betriebssystem genutzt wird, ist dies auch die System-Testumgebung. Hier wird im Druckmanager eine Testseite für den Laserdrucker angefordert. Der Thermodrucker wird über eine von Epson mitgelieferte Software getestet. Es handelt sich um eine Demo Applikation, die es ermöglicht die richtige Konfiguration und die Funktionstüchtigkeit zu ermittelt bzw. sicher zu stellen. Sind diese Tests erfolgreich, ist sichergestellt, dass die Hardwarefunktionen vorhanden sind.

Softwareseitig müssen folgende Komponenten installiert bzw. vorhanden sein:

- Java/Java Libraries (JRE-6u12, epos, JPS, JAI)
- SQL Datenbank
- Datenbankmodell mit Initialdaten

- Kassensoftware (Schilderservice.jar)

Im folgenden wird nun ein Testfallablauf beschrieben der einen einfachen Kassiervorgang darstellt:

1. Start des Kassenprogramms (java -jar Schilderservice.jar)  
Der Aufruf erfolgt aus der Konsole, um mögliche Exceptions sehen zu können.  
**Erwartet:** Bild mit Logo und der Information des Ladevorganges. Nach beenden des Ladevorganges ein *Startbutton* in Form des Logos.
2. Betätigen des *Startbutton*  
**Erwartet:** Kassenstart Login. Abfrage nach gültiger PIN.
3. PIN Eingabe (Eingabe: "1000")  
**Erwartet:** Fehlermeldung: Das eingegebene Passwort ist falsch!
4. PIN Eingabe (Eingabe: "1")  
**Erwartet:** Kasse gestartet und erwartet die Eingabe für einen Kassiervorgang.
5. Auswählen eines Artikel mittels Button (520x110)  
**Erwartet:** Artikel erscheint in der Artikelliste.
6. Bezahlvorgang auslösen und bestätigen.  
**Erwartet:** Aufforderung zur Eingabe des Gesamtpreises, durch die sofort Bestätigung, Annahme des genau gegebenen Betrages. Bereit zum nächsten Kassiervorgang.
7. Quittungsdruck des getätigten Kassiervorganges aufrufen.  
**Erwartet:** Thermodrucker gibt Quittung aus.

Nach erfolgreichem Testdurchlauf ist eine aktive Verbindung zur Datenbank gewährleistet, das Datenbankmodul schreibt und liest erfolgreich aus der Datenbank und der Thermodrucker gibt Quittungen aus. Für den Laserdrucker und alle anderen Komponenten von und zur Datenbank sind ähnliche Testdurchläufe erfolgt.

## 7. Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Design einer Kassensoftware und deren Implementierung zum Einsatz auf spezifischer Kassenhardware in einer Kennzeichenprägestelle entwickelt.

Zu Beginn wird die Anforderungserhebung erstellt, diese liefert die Grundlage zur Erstellung des Designs. Hier wird im ersten Schritt die Anforderung an die Hard- und Software analysiert. Die Hardware hat auch unter Dauerbelastung und schlechten Umgebungseinflüssen ordnungsgemäß zu funktionieren. Die Kassensoftware hat viele Aufgaben zu erfüllen, was das Design und die Implementierung erschwert. Die Fehlertoleranz muss trotz der schwierigen Bedingungen, die an die Hard- und Software gestellt werden, hoch sein. Im nächsten Schritt ist eine Spezifizierung der Ziele und Anforderungen des Auftraggebers, sowie der Nutzen des Anwenders aufgestellt. Die funktionalen Anforderungen sind nach Funktionsaspekten gegliedert, die nicht funktionalen Anforderungen nach Randbedingungen, Performanz und Ergonomie. Die zu erwartenden Kosten für die Entwicklung, Wartung/Erweiterung zeigen, dass es sehr schwer ist eine Abschätzung der zu erwartenden Kosten für die Entwicklung zu erstellen. Für die Wartung bzw. Erweiterung gilt ähnliches, da dies sehr von den Anforderungen abhängig ist.

Das Design der Software ist auf eine kompakte und wartbare Implementierung ausgerichtet. Die Datenbank bildet die zentrale Rolle zur Datenvorhaltung. Alle Vorgänge werden unmittelbar in der Datenbank festgehalten. Dies garantiert einen immer aktuellen Datenstand. Zur Sicherung ist eine zweite dezentrale Datenbank an das System gekoppelt. Dadurch wird ermöglicht, dass bei einem Defekt der Hardware, eine Kopie der Geschäftsvorgänge vorhanden ist. Ausserdem ist es dadurch möglich, auch wenn man nicht an dem Kassensystem arbeitet oder dieses ausgeschaltet ist, eine Einsicht in die Geschäftsvorgänge zu bekommen. Bei der Implementierung wurde auf eine einfache und intuitive Bedienoberfläche Wert gelegt. So ist es möglich die Kassensoftware per Touchscreen zu bedienen und es muss nur selten auf Eingabegeräte, wie Maus und Tastatur zurückgegriffen werden. Die Implementierung nutzt die standardisierte POS Schnittstelle der Firma Epson zur Ansteuerung des Thermodruckers. Die Möglichkeit zur einfachen Erweiterung eines Linedisplays ist vorgesehen, hier wird die POS Schnittstelle der Firma WINCOR benötigt. Die Anbindung an das Datenbanksystem erfolgt über die JDBC Treiber, die es ermöglichen Java Programme an verschiedene Datenbanksysteme zu koppeln. Hier wird der MySQL Datenbanktreiber[10] geladen und für die Anbindung genutzt.

Das eingesetzte Testverfahren mittels JUnit, half während der Entwicklung die Fehler frühzeitig zu entdecken, um diese zu korrigieren. Der Systemtest zeigte die Funktionalität der entwickelten Kassensoftware auf dem Zielsystem.

## Ausblick

Während der Entwicklung ist es zu einer Änderung im Zusammenhang mit der externen Datenbank gekommen. Durch einen Serverwechsel ist die Möglichkeit der direkten Anbindung mittels JDBC entfallen. Hierzu muss eine Lösung gefunden werden, ein Ansatz ist entwickelt und dem Anhang beigefügt. Eine Integration in das Kassensystem steht zur Zeit noch aus.

Änderungen während der Entwicklung bereiten oft Probleme. Durch ein strukturiertes Design unter zu Hilfenahme von passenden Design Pattern, können Änderungen die nicht grundlegende Veränderungen herbei führen gut abfangen werden. So auch bei diesem Problem, es muss lediglich ein weiteres Datenbankmodul erstellt werden. Dieses ersetzt dann, das bisherige Modul zur Anbindung an die externe Datenbank.

# Literaturverzeichnis

- [1] BUTH, Bettina: *Software Engineering 2*, Hochschule für Angewandte Wissenschaften Hamburg, Vorlesungsunterlagen, 2009
- [2] ECLIPSE FOUNDATION: Eclipse IDE for Java Developers. (2011). – URL <http://www.eclipse.org/downloads/packages/release/helios/sr2>. – Abruf: 2. April 2011
- [3] EILEBRECHT, Karl ; STARKE, Gernot: *Patterns kompakt Entwurfsmuster für effektive Software-Entwicklung*. 3. Heidelberg : Spektrum, 2010. – ISBN 978-3-8274-2525-6
- [4] HANSER, Eckhart: *Agile Prozesse: Von XP über Scrum bis MAP*. 0. Berlin : Springer Verlag, 2010. – ISBN 978-3-642-12313-9
- [5] HRUSCHKA, Peter ; RUPP, Chris ; STARKE, Gernot: *Agility kompakt Tipps für erfolgreiche Systementwicklung*. 2. Heidelberg : Spektrum, 2009. – ISBN 978-3-8274-2204-0
- [6] JUNIT.ORG: JUnit. (2011). – URL <http://www.junit.org/>. – Abruf: 2. Juli 2011
- [7] KASSEN.NET: LaCash® Einzelhandel. (2005). – URL [http://kassen.net/datenblaetter/31\\_1109858600\\_LaCashHandel.pdf](http://kassen.net/datenblaetter/31_1109858600_LaCashHandel.pdf). – Abruf: 20. August 2011
- [8] MICROSOFT: Visio Standard 2010. (2011). – URL <http://emea.microsoftstore.com/DE/de-DE/Microsoft/Visio-Standard-2010>. – Abruf: 1. Mai 2011
- [9] MICROSYSTEMS, Sun: Java BluePrints Model-View-Controller. (2002). – URL <http://java.sun.com/blueprints/patterns/MVC-detailed.html>. – Abruf: 5. Juli 2011
- [10] MYSQL: Connector/J 5.1.17. (2011). – URL <http://dev.mysql.com/downloads/connector/j/>. – Abruf: 10. Juli 2011
- [11] NATIONAL RETAIL FEDERATION: Unified POS Frequently Asked Questions. (2011). – URL <http://www.nrf-arts.org/content/unified-pos-frequently-asked-questions>. – Abruf: 15. April 2011
- [12] NATIONAL RETAIL FEDERATION: UnifiedPOS. (2011). – URL <http://www.nrf-arts.org/content/unifiedpos>. – Abruf: 15. April 2011

- [13] NETBEANS: NetBeans IDE 7.0. (2011). – URL <http://netbeans.org/downloads/7.0/index.html>. – Abruf: 2. April 2011
- [14] OMONDO: EclipseUML. (2011). – URL <http://www.ejb3.org/>. – Abruf: 1. Mai 2011
- [15] PETTERS, Hartmut: Software Engineering Informatik II. (2003). – URL <http://www.petters.eu/p2-Dateien/SE%20K2%20SW-Prozess.pdf>. – Abruf: 12. April 2011
- [16] POSBILL.COM: PosBill<sup>®</sup>Picture. (2010). – URL <http://www.posbill.com/typo3temp/pics/15286568ab.jpg>. – Abruf: 20. August 2011
- [17] RICOBANDITO ; ALAIBOT ; RICHARDVERYARD ; MAREK 69 ; LOTJE ; MTC-PAUL: UnifiedPOS. (2011). – URL <http://en.wikipedia.org/wiki/UnifiedPOS>. – Abruf: 10. April 2011
- [18] SCHILL, Michael: Test-Driven Development. (2005). – URL <http://www.spies.informatik.tu-muenchen.de/lehre/seminare/SS05/hauptsem/Ausarbeitung02.pdf>. – Abruf: 10. Juli 2011
- [19] SSDOFFICE.DE: Kassensystem Einzelhandel 2008. (2008). – URL [http://www.sds-office.de/index.php/de/Kassensystem\\_Einzelhandel\\_2008\\_Software/1\\_KAT150](http://www.sds-office.de/index.php/de/Kassensystem_Einzelhandel_2008_Software/1_KAT150). – Abruf: 20. August 2011
- [20] THE OMNI GROUP: VOmniGrafle 5. (2011). – URL <http://www.omnigroup.com/products/omnigraffle/>. – Abruf: 3. Mai 2011
- [21] ULLENBOOM, Christian: Java ist auch eine Insel. (2006). ISBN 978-3-89842-747-0
- [22] USER, Wiki: Agile Softwareentwicklung. (2011). – URL [http://de.wikipedia.org/wiki/Agile\\_Softwareentwicklung](http://de.wikipedia.org/wiki/Agile_Softwareentwicklung). – Abruf: 9. September 2011
- [23] VECTRON: POS Systems. (2011). – URL <http://www.vectron.de/?l=de>. – Abruf: 15. Mai 2011
- [24] VISUAL PARADIGM: Visual Paradigm for UML 8.3 Community Edition. (2011). – URL <http://www.visual-paradigm.com/download/vpuml.jsp?edition=ce>. – Abruf: 2. Mai 2011
- [25] WINCOR NIXDORF: UnifiedPOS. (2006). – URL [http://www.wincor-nixdorf.com/internet/cae/servlet/contentblob/70238/publicationFile/6401/PresentationUnifiedPOS\\_download.pdf](http://www.wincor-nixdorf.com/internet/cae/servlet/contentblob/70238/publicationFile/6401/PresentationUnifiedPOS_download.pdf). – Abruf: 10. Mai 2011



- 
- [26] WINCOR NIXDORF: Kassenfamilie BEETLE BEETLE /M-II. (2009).  
– URL <http://www.wincor-nixdorf.com/internet/cae/servlet/contentblob/34100/publicationFile/59891/DatasheetBeetleMII.pdf>. – Abruf: 15. August 2011
- [27] WINCOR NIXDORF: POS Systems. (2011). – URL [http://www.wincor-nixdorf.com/internet/site\\_EN/EN/Products/Hardware/POSSystems/Node.html](http://www.wincor-nixdorf.com/internet/site_EN/EN/Products/Hardware/POSSystems/Node.html). – Abruf: 15. Mai 2011
- [28] WOLF, Henning ; BLEEK, Wolf-Gideon: *Agile Softwareentwicklung: Werte, Konzepte und Methoden*. 2. Wiesbaden : dpunkt.verlag, 2010. – ISBN 978-3-89864-701-4

# Abbildungsverzeichnis

1.1.	Berührungsempfindlicher Bildschirm [26] . . . . .	3
1.2.	Recheneinheit mit Betriebssystem und POS-Software [26] . . . . .	3
1.3.	Kundenanzeige für Informationen über den Verkaufsvorgang (z.B. Preisangabe) [26] . . . . .	3
1.4.	Kassenslade zum sicheren verwahren des Geldes [26] . . . . .	3
1.5.	Handscanner zum erfassen von Strichcodes [26] . . . . .	3
1.6.	Kassendrucker zum erstellen von Kundenquittungen [26] . . . . .	3
2.1.	UnifiedPOS unterstütze Gerätekategorien [11] . . . . .	7
2.2.	JavaPOS Architektur [25] . . . . .	7
2.3.	MVC-Pattern Konzept . . . . .	12
3.1.	Schematische Darstellung der GUI . . . . .	15
3.2.	Kassensoftware auf dem Markt . . . . .	16
3.3.	GUI PosBill® [16] . . . . .	16
3.4.	GUI LaCash® [7] . . . . .	17
3.5.	GUI Kassensoftware Einzelhandel 2008 [19] . . . . .	17
4.1.	UI Visual Paradigm [24] . . . . .	24
4.2.	UI OmniGrafle 5 [20] . . . . .	24
4.3.	UML Tools (Visual Paradigm [24]; OmniGrafle 5 [20]; EclipseUML [14]; Visio [8]) . . . . .	25
4.4.	Entity-Relationship-Modell . . . . .	26
4.5.	Auszug Klassendiagramm – Start . . . . .	27
4.6.	Entityklasse – Artikel . . . . .	27
4.7.	Druckklasse – Modul zum druck auf den Standarddrucker . . . . .	27
4.8.	Auszug Klassendiagramm – View, Datenbank und Verwaltung . . . . .	28
4.9.	Sequenzdiagramm Kassenstart – Kassiervorgang . . . . .	31
4.10.	Datenbankmodell . . . . .	33
5.1.	Auswahl an Entwicklungsumgebungen . . . . .	34
5.2.	Eclipse JDT User Interface . . . . .	35
5.3.	Netbeans IDE User Interface . . . . .	36
5.4.	Package Struktur . . . . .	37
5.5.	Liste der generierbaren Button Events . . . . .	39
5.6.	Action Task Menü . . . . .	39

---

A.1. Sequenzdiagramm zur Kommunikation zwischen der Java Applikation und  
der Datenbank über den PHP Server . . . . . 56

# A. Datenbankbindung mittels PHP Server

Die Anbindung eines Java Projekts über JDBC-Treiber zu einer Datenbank, die bei einem Webhoster gelegen ist, erweist sich als nicht möglich, wenn der Webhoster seine datenbankserver gegen externe Verbindungen schützt. Es können nur solche Verbindungen zur Datenbank aufgebaut werden, die über den Webhostserver angefordert werden.

Zur Lösung des Problems gibt es nur wenige Möglichkeiten, eine wäre den Webhoster zu wechseln, der seine Datenbank nicht nach aussen absichert. Eine weitere wäre ein Webhosting-Paket zu wählen, welches erlaubt eigene Applikationen auf dem Datenbankserver laufen zu lassen und somit eine Kommunikation herzustellen. Die im folgenden vorgestellte Lösung benötigt weder einen Wechsel des Webhosters, noch ein Upgrade auf ein anderes Paket. Die entwickelte Lösung nutzt den PHP Server des Webhosters als Kommunikationsschnittstelle zwischen der Java Applikation und der externen Datenbank.

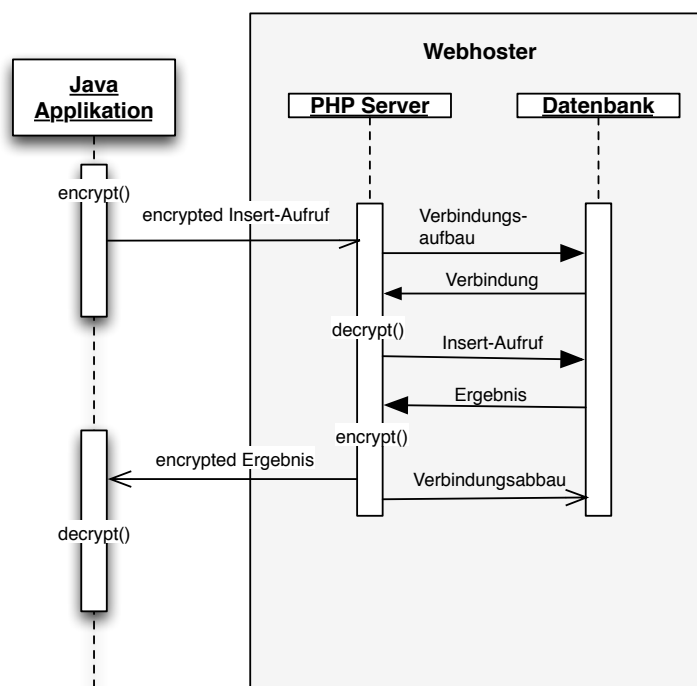


Abb. A.1.: Sequenzdiagramm zur Kommunikation zwischen der Java Applikation und der Datenbank über den PHP Server

In der Abbildung [A.1](#) ist der Kommunikationsablauf zwischen der Java Applikation und der externen Datenbank als Sequenzdiagramm zu sehen. Die verschlüsselten Daten samt Zeitstempel werden an das PHP-Skript auf dem PHP-Server übertragen. Der PHP-Server baut eine Verbindung zur Datenbank auf. Steht die Verbindung zur Datenbank, werden die empfangenen Daten entschlüsselt und der Zeitstempel überprüft. Ist dieser korrekt, wird der zuzuschickende SQL-Aufruf an die Datenbank geschickt. Das Ergebnis der Datenbank wird verschlüsselt und an die Java Applikation zurück gesendet. Diese kann nun die Daten entschlüsseln und mit den Daten weiter arbeiten.

In dem Listing [A.1](#) sind die Methoden zur Ver- und Entschlüsselung von zu sendenden Daten dargestellt. Die Daten die zum PHP-Server gehen, werden zuerst mit einem AES Verschlüsselungsverfahren verschlüsselt. Da die Übertragung zum PHP-Server mittels HTTP erfolgt, ist der Einsatz von beliebigen Zeichen nicht möglich, daher werden die verschlüsselten Daten in das Base64 Format kodiert.

Werden Daten empfangen werden diese mit der *decrypt()* Methode entschlüsselt. Hier wird zuerst das Base64 Format dekodiert und darauf folgend das AES Entschlüsselungsverfahren verwendet.

Das Ver- und Entschlüsselungsverfahren beruht auf einem Startparameter und dem eigentlichen geheimen Schlüssel. Diese beiden Werte müssen der Java Applikation und dem PHP-Skript bekannt sein.

Das Listing [A.2](#) zeigt die Methoden, die das Senden und Empfangen der Daten zum und vom PHP-Server übernehmen. Die senden Methode erwartet eine Liste von Query-Strings. Wobei der erste Listeneintrag ein aktueller Zeitstempel sein muss, der verschlüsselt wird. Alle weiteren Listeneinträge werden ebenfalls verschlüsselt und dem Zeitstempel angehängt. Diese Daten werden mittels URLConnection zu dem PHP-Server und dem darauf liegenden PHP-Skript geschickt.

Das PHP-Skript, welches die Daten empfängt, ist in dem Listing [A.3](#) zu sehen. Das PHP Skript wird durch den PHP-Server verarbeitet und baut zunächst eine Verbindung zur Datenbank auf. Steht die Verbindung zur Datenbank, wird der verschlüsselte Zeitstempel entschlüsselt und es wird geprüft, ob sich der Zeitstempel in dem vorgegebenen Zeitfenster befindet. Ist dies der Fall, werden die restlichen Daten entschlüsselt und die darin befindlichen Befehle an die Datenbank weitergeleitet. Das Ergebnis von der Datenbank wird wieder verschlüsselt und als Antwort zurückgegeben. Abschließend wird die Verbindung zur Datenbank beendet.

```

0  /**
   * PhpPostConnect.java
   * Diese Klasse baut eine Verbindung zu einer PHP-Seite auf
   * um Daten ueber den "POST" Command zu senden und Daten
   * die die Seite mit einem "echo" sendet zu empfangen
   * @author Can Eskikaya
   */
5  public class PhpPostConnect {

10     /** Beinhaltet die Adresse der PHP-Seite */
    private URL sitepath;
    /** Die Verbindung zur URL */
    private URLConnection con;

15     /**
     * Leerer Konstruktor, vor dem Senden und Empfangen von Daten
     * muss die Ziel URL angegeben werden.
     */
    public PhpPostConnect() {
20     }

    /**
     * Konstruktor mit Ziel URL
     * @param sitepath Die URL der PHP-Seite
     */
25     public PhpPostConnect(URL sitepath) {
        this.sitepath = sitepath;
    }

30     /**
     * Zum verschluesseln der Nachricht. Erst wird mit einem AES Verschlusselungsverfahren verschluesselt
     * danach in Base64 gewandelt.
     * @param data Der String der verschluesselt werden soll
     */
    public static String encrypt(String data) {
35        String encryptedData = null;
        try {
            SecretKeySpec keySpec = new SecretKeySpec("oldesloeoldesloe".getBytes(), "AES");
            IvParameterSpec initialVector = new IvParameterSpec("1234567812345678".getBytes());
            Cipher cipher = Cipher.getInstance("AES/CFB8/NoPadding");
            cipher.init(Cipher.ENCRYPT_MODE, keySpec, initialVector);
            byte[] decryptedByteArray = (data.getBytes());
            byte[] encryptedByteArray = cipher.doFinal(decryptedByteArray);
            byte[] encode = (new org.apache.commons.codec.binary.Base64()).encode(encryptedByteArray);
            encryptedData = new String(encode, "UTF8");
45        } catch (Exception e) {
            e.printStackTrace();
        }
        return encryptedData;
    }

50     /**
     * Zum entschluesseln der Nachricht. Es wird erst Base64,
     * danach mit dem AES Verschlusselungsverfahren entschluesselt.
     * @param encryptedData Der String der entschluesselt werden soll
     */
    private String decrypt(String encryptedData) {
55        String decryptedData = null;
        try {
            SecretKeySpec keySpec = new SecretKeySpec("oldesloeoldesloe".getBytes(), "AES");
            IvParameterSpec initialVector = new IvParameterSpec("1234567812345678".getBytes());
            Cipher cipher = Cipher.getInstance("AES/CFB8/NoPadding");
            cipher.init(Cipher.DECRYPT_MODE, keySpec, initialVector);
            byte[] encryptedByteArray = (new org.apache.commons.codec.binary.Base64()).decode(encryptedData.getBytes());
            byte[] decryptedByteArray = cipher.doFinal(encryptedByteArray);
            decryptedData = new String(decryptedByteArray, "UTF8");
65        } catch (Exception e) {
            e.printStackTrace();
        }
        return decryptedData;
    }

70     /**
     * Setzt die URL von der PHP-Seite
     * @param sitepath Die URL zur PHP-Seite
     */
    public void setSitePath(URL sitepath) {
75        this.sitepath = sitepath;
    }

80     /**
     * Zum bekommen der URL der PHP-Seite
     * @return Die URL der PHP-Seite
     */
    public URL getSitePath() {
85        return this.sitepath;
    }
}

```

Listing A.1: Ver- und Entschlüsselung zur Kommunikation mit dem PHP Server

```

0  /**
   * Nimmt die Daten zum senden entgegen und bringt Sie in das richtige Format,
   * danach wird gesendet.
   * @param dataList Die Liste von Strings die gesendet werden soll
   */
5  public void send(List<String> dataList) {
   if (sitepath==null){
   return;
   }
   if (dataList.isEmpty()) {
10    return;
   }
   String toSend = encrypt(String.valueOf(System.currentTimeMillis()).substring(0, 10)).replace("=", "#") + "=" +
   encrypt(dataList.get(0));
   dataList.remove(0);

15    if (dataList.isEmpty()) {
   try {
   send(toSend.getBytes());
   } catch (IOException ex) {
20     ex.printStackTrace();
   }
   return;
   }

   int count = 0;

25    for (String data : dataList) {
   toSend.concat("&" + encrypt(count + "").replace("=", "#") + "=" + encrypt(data));
   count++;
   }
30    if (dataList.isEmpty()) {
   try {
   send(toSend.getBytes());
   } catch (IOException ex) {
35     ex.printStackTrace();
   }
   }
   }

40  /**
   * Sendet Daten zu der PHP-Seite
   * @param data Die Daten, die gesendet werden sollen
   * @throws IOException
   */
45  private void send(byte[] data) throws IOException {
   con = null;
   if (con == null) {
   con = sitepath.openConnection();
   }
50    if (con.getDoOutput() == false) {
   con.setDoOutput(true);
   }
   OutputStream out = con.getOutputStream();
   out.write(data);
   out.flush();
55  }

60  /**
   * Liest Daten die von der PHP-Seite empfangen wurden
   * @return Die empfangenen Daten
   * @throws IOException
   */
65  public String read() throws IOException {
   if (con == null) {
   con = sitepath.openConnection();
   }
   InputStream in = con.getInputStream();
   int c = 0;
   StringBuilder incoming = new StringBuilder();
70    while (c >= 0) {
   c = in.read();
   if ((int) c == -1) {
   break;
   }
   incoming.append((char) c);
75  }
   }
   }
}

```

Listing A.2: Senden und empfangen der Daten zum und vom PHP Server

```

0 <?php
  header("Content-type: _text/html; _charset=utf-8");

  function encrypt($message) {
    return base64_encode(
5      mcrypt_encrypt(
        MCRYPT_RIJNDAEL_128,
        "oldesloeoldesloe",
        $message,
        MCRYPT_MODE_CFB,
        "1234567812345678"
10      )
    );
  }

  function decrypt($message) {
    $messageDecode = base64_decode($message);
    //echo $messageDecode
    return
15      mcrypt_decrypt(
        MCRYPT_RIJNDAEL_128,
        "oldesloeoldesloe",
        $messageDecode,
        MCRYPT_MODE_CFB,
        "1234567812345678"
20      );
  }

  $server= "*****.de";           /* Adresse des Datenbankservers */
  $user= "DatenbankBenutzerName"; /* Datenbank-Benutzername */
  $password= "password";         /* Passowrt */
30  $datenbank= "NameDerDatenbank"; /* Name der Datenbank */
  $tabelle= "";                  /* Name der Tabelle, kann frei gewachlt werden */
  $result= "";
  $value3= "";

35  /* Zugriff auf SQL-Server */
  MYSQL_CONNECT($server, $user, $password) or die("<H3>Datenbankserver_nicht_erreichbar </H3>");
  MYSQL_SELECT_DB($datenbank) or die("Datenbank_nicht_vorhanden#");
  mysql_query("SET NAMES _utf8 _");
  mysql_query("SET CHARACTER_SET _utf8 _");

40  if(strtolower($_SERVER['REQUEST_METHOD']) == 'post') {

    $_firstkey = array_keys($_POST);

45    /*
     * Das "="-Zeichen ist fuer Key=Value vorgesehen.
     * Bei der Verschlusselung kommt es als Zeichen vor und ist fuer die uebertragung ersetzt worden durch eine "#".
     * Zum entschluesseln muss dies wieder rueckgaengig gemacht werden.
     */
    $encryptedKey=str_replace("#","=", $_firstkey[0]);
    /*
50    * Der $key ist ein Zeitstempel. long Time millis wobei nur die ersten 10 Zahlen genutzt werden.
     * Beim Abfangen (man in the middle) der uebertragung sind die Datenpakete kein 2 tes mal nutzbar, da der
     * Zeitstempel stimmen muss.
     */
55    $decryptKey=decrypt($encryptedKey);

    // Zeitstempel wird auf gueltigkeit geprueft
60    if(( $decryptKey-10) <= (int)gmdate('U') && (int)gmdate('U')<= ($decryptKey+10)){

      $res=0;

65      foreach($_POST as $key => $value) {

        /*
         * "+"-Zeichen werden als leerzeichen interpretiert, daher ist der austausch der leerzeichen
         * durch "+"
         * erforderlich. Zum decrypten
         */
70        $query=str_replace("_", "+", "$value");

        //Der query wird entschlueaesselt
        $decryptText = decrypt($query);

75        $res = MYSQL_QUERY($decryptText) or die("Fehler:_ . mysql_error());
      }
      echo encrypt($res);
    }
  }

  MYSQL_CLOSE();
?>

```

Listing A.3: PHP Serverskript



# Abkürzungsverzeichnis

<b>Abb.</b> .....	Abbildung
<b>API</b> .....	Application Programming Interface
<b>ASD</b> .....	Adaptive Software Development
<b>bzw.</b> .....	beziehungsweise
<b>CAT</b> .....	Credit Authorization Terminal
<b>CDDL</b> .....	Common Development and Distribution License
<b>COM</b> .....	Component Object Model
<b>DPD</b> .....	Dynamic Parcel Distribution
<b>DSDM</b> .....	Dynamic System Development Method
<b>EC</b> .....	Electronic Cash
<b>EIA</b> .....	Electronic Industries Alliance
<b>epos</b> .....	Epson Point of Service
<b>ER</b> .....	Entity-Relationship
<b>F-</b> .....	funktional
<b>FDD</b> .....	Feature Driven Development
<b>FN-</b> .....	nicht funktional
<b>FUN</b> .....	Spass
<b>ggf.</b> .....	gegebenenfalls
<b>GNU</b> .....	General Public License
<b>GPL2</b> .....	General Public License 2
<b>GUI</b> .....	Graphical User Interface
<b>ID</b> .....	Identifikator
<b>IDE</b> .....	Integrated development environment
<b>IDL</b> .....	Interface Description Language
<b>JAI</b> .....	Java Advanced Imaging
<b>JCL</b> .....	Java Compiled Language
<b>JDBC</b> .....	Java Database Connectivity
<b>JDT</b> .....	Java Development Tools
<b>JPEG</b> .....	CCITT Recommendation T.81 - Joint Photographic Experts Group
<b>JPOS</b> .....	Java Point of Service
<b>JPOS Registry</b> .	Java Point of Service Registry
<b>JPS</b> .....	Java Print Service
<b>JRE</b> .....	Java Runtime Environment
<b>KFZ</b> .....	Kraftfahrzeug
<b>KZ</b> .....	Kurzzeit
<b>LAN</b> .....	Local Area Network
<b>m</b> .....	minimal 1 bis unendlich
<b>max.</b> .....	Maximal
<b>mc</b> .....	minimal 0 bis unendlich
<b>MICR</b> .....	Magnetic Ink Character Recognition
<b>MMI</b> .....	Mensch Maschine Interface
<b>MR</b> .....	Monatsrechnung
<b>MwSt.</b> .....	Mehrwertsteuer
<b>Nr</b> .....	Nummer

---

<b>OLE</b> .....	Object Linking and Embedding
<b>OO</b> .....	Objektorientierung
<b>OPOS</b> .....	OLE for Retail POS
<b>OS</b> .....	Operating System
<b>PC</b> .....	Personal Computer
<b>PDF</b> .....	Portable Document Format
<b>PIN</b> .....	Persönliche Identifikationsnummer
<b>POS</b> .....	Point of Service
<b>RFID</b> .....	Radio-Frequency Identification
<b>SQL</b> .....	Structured Query Language
<b>TDD</b> .....	Test Driven Development
<b>UDD</b> .....	Usability Driven Development
<b>UI</b> .....	User Interface
<b>UML</b> .....	Unified Modeling Language
<b>UPOS</b> .....	Unified point of Service
<b>vgl.</b> .....	vergleiche
<b>WaWi</b> .....	Warenwirtschaft
<b>XP</b> .....	Extreme Programming
<b>z. B.</b> .....	zum Beispiel
<b>ÜW</b> .....	Überweisung

**CD: Kassensoftware Code, Treiber,  
Datenbank**

# Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 22. September 2011

Ort, Datum

Unterschrift