|  |
| --- |
| IT14B |
| Dokumentation |
| Inventory Management System |
|  |
| **Hagen Frind, Henry Martens, Dominik Traise** |
| **18.05.2016** |

|  |
| --- |
| Desktopanwendung zur Verwaltung von Hardware-Komponenten in einem Lager. |

# Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc450841422)

[Abkürzungsverzeichnis A](#_Toc450841423)

[1. Einleitung 1](#_Toc450841424)

[1.1. Projektziel 1](#_Toc450841425)

[1.2. Projektumfeld 1](#_Toc450841426)

[1.3. Projektbegründung 1](#_Toc450841427)

[1.4. Projektschnittstellen 1](#_Toc450841428)

[1.5. Projektabgrenzung 1](#_Toc450841429)

[2. Projektplanung 2](#_Toc450841430)

[2.1. Projektzeitplan 2](#_Toc450841431)

[2.2. Ressourcenplanung 2](#_Toc450841432)

[2.3. Entwicklungsprozess 2](#_Toc450841433)

[2.3.1. Spiralmodell 2](#_Toc450841434)

[2.3.2. Test-Driven-Development 3](#_Toc450841435)

[3. Analysephase 3](#_Toc450841436)

[3.1. Ist-Analyse 3](#_Toc450841437)

[3.2. Nicht-monetäre Vorteile 3](#_Toc450841438)

[3.3. Lastenheft 3](#_Toc450841439)

[3.4. Zwischenstand 3](#_Toc450841440)

[4. Entwurfsphase 4](#_Toc450841441)

[4.1. Zielplattform 4](#_Toc450841442)

[4.2. Architekturdesign 4](#_Toc450841443)

[4.3. Benutzeroberfläche 4](#_Toc450841444)

[4.4. Datenbankentwurf 4](#_Toc450841445)

[4.5. Geschäftslogik 4](#_Toc450841446)

[4.6. Pflichtenheft 5](#_Toc450841447)

[4.7. Zwischenstand 5](#_Toc450841448)

[5. Implementierungsphase 5](#_Toc450841449)

[5.1. Iterationsplan 5](#_Toc450841450)

[5.2. Implementierung der Datenstrukturen 5](#_Toc450841451)

[5.3. Implementierung der Geschäftslogik 5](#_Toc450841452)

[5.4. Implementierung der Benutzeroberfläche 6](#_Toc450841453)

[5.5. Zwischenstand 6](#_Toc450841454)

[6. Abnahmephase 6](#_Toc450841455)

[6.1. Zwischenstand 6](#_Toc450841456)

[7. Dokumentation 6](#_Toc450841457)

[7.1. Zwischenstand 6](#_Toc450841458)

[8. Fazit 7](#_Toc450841459)

[8.1. Soll-/Ist-Vergleich 7](#_Toc450841460)

[8.2. Aus dem Projekt Gewonnenes 7](#_Toc450841461)

[8.3. Ausblick 7](#_Toc450841462)

[A. Anhang I](#_Toc450841463)

[A.1 Detaillierter Zeitplan I](#_Toc450841464)

[A.2 Ressourcen I](#_Toc450841465)

[A.3 Lastenheft II](#_Toc450841466)

[A.4 Schichtenmodell III](#_Toc450841467)

[A.5 Oberflächenentwürfe IV](#_Toc450841468)

[A.6 ER-Modell V](#_Toc450841469)

[A.7 Struktogramm VI](#_Toc450841470)

[A.8 Pflichtenheft VI](#_Toc450841471)

[A.9 Iterationsplan VII](#_Toc450841472)

[A.10 Quellcode VIII](#_Toc450841473)

[A.11 Screenshot der Anwendung IX](#_Toc450841474)

# ****Abkürzungsverzeichnis****

**BBS** Berufsbildende Schule  
**CSV** Comma Separated Value  
**DB** Datenbank  
**ERM** Entity-Relationship-Modell  
**GUI** Graphical User Interface  
**IMS** Inventory Management System  
**SQL** Structured Query Language  
**TDD** Test Driven Development  
**WPF** Windows Presentation Foundation  
**XAML** Extensible-Application-Markup-Language

# Einleitung

Die folgende Projektdokumentation schildert den Ablauf des Datenbankprojekts, welches das Projektteam im Rahmen einer ihnen gestellten Aufgabe im Lernfeld 6 der [BBS](#_Abkürzungsverzeichnis) 1 in Lüneburg durchgeführt hat.

## Projektziel

Ziel des Projektes ist eine erleichterte Erfassung und Verwaltung von verschiedenen Hardware-Komponenten. Dazu soll eine Datenbank erstellt werden, welche die Daten dauerhaft in einem persistenten Zustand speichert, zudem soll eine graphische Benutzeroberfläche ([GUI](#_Abkürzungsverzeichnis)) entworfen werden über welche dem Nutzer eine einfache Verwaltung der Komponenten ermöglicht wird. Die Anwendung soll das Auslesen und Sortieren der Teile erlauben, sodass ein besserer Überblick über den aktuellen Lagerbestand entsteht. Durch das System können menschliche Fehler, wie zum Beispiel das Vergessen von wichtigen Informationen, vermieden werden.

## Projektumfeld

Die zu erstellende Software wird überwiegend von den Mitarbeitern des Lagers genutzt werden, da diese für die Verwaltung der Lagerbestände zuständig sind. Die Ein- und Verkaufsmitarbeiter stellen eine weitere Gruppe von Benutzern da, diese benötigen einen Überblick über die im Lager abgelegten Komponenten um zielführende Verhandlungen mit Kunden oder Händlern führen zu können.

## Projektbegründung

In einem Lager gibt es oftmals eine große Anzahl an Hardware-Bauteilen, welche ohne eine Struktur und die entsprechenden Dokumente nur schwer zu Verwalten sind. So müssen die Mitarbeiter beispielsweise Tabellen und Listen führen, um den Überblick über den aktuellen Lagerbestand zu behalten. Diese manuelle Verwaltung kostet viel Zeit und kann im schlimmsten Fall den Arbeitsfluss an mehreren Stellen verzögern oder sogar unterbrechen. Aus diesen Gründen soll eine Software erstellt werden, welche die Verwaltung der Komponenten erleichtert und somit langfristig eine Kostenersparnis bewirken kann.

## Projektschnittstellen

Zu Beginn des Projektes wurden die späteren Schnittstellen zur Anwendung definiert. Die Anwendung soll mit einer Datenbank kommunizieren, welche auf einem externen System betrieben werden kann, diese kann mittels einer grafischen Benutzeroberfläche auf den Client-Rechnern von den Lagerangestellten verwaltet werden. Auf den Client-Rechnern wird ein Windows-System betrieben. Das zu erstellende System wird von Mitarbeitern des Ein- und Verkaufs, sowie den Lagerangestellten genutzt werden.

## Projektabgrenzung

Da das Team bei der Durchführung des Projektes auf eine Zeit von 24 Stunden begrenzt war, wurde die Möglichkeit der Filterung von Hardware-Komponenten anhand ihrer Attribute bewusst nicht in die Planung des Projektes integriert.

# Projektplanung

Während der Projektplanung wurden der Ablauf, sowie die zeitliche Durchführung des Projektes festgelegt. Zudem wurden die benötigten Ressourcen geplant, welche im Laufe der Projektdurchführung genutzt wurden.

## Projektzeitplan

Insgesamt standen dem Projektteam für die Umsetzung des Projektes 24 Stunden zur Verfügung. Die Stunden wurden auf verschiedene Phasen aufgeteilt, welche wiederrum die detaillierten Aufgaben enthielten. Die definierten Aufgaben des Projektes wurden je nach Kenntnisstand und Erfahrung unter den Teammitgliedern aufgeteilt. Die Zeitplanung mit einer Unterteilung nach Phasen kann der Tabelle 1: Grober Zeitplan entnommen werden. Die detaillierte Planung findet sich im Anhang unter [A.1 Detaillierter Zeitplan](#_Detaillierter_Zeitplan).

|  |  |
| --- | --- |
| Projektphase | Geplante Zeit |
| Analysephase | 2 Stunden |
| Entwurfsphase | 4 Stunden |
| Implementierungsphase | 13 Stunden |
| Abnahmetest der Gruppe | 1 Stunde |
| Erstellen der Dokumentation | 4 Stunden |
| Gesamt | 24 Stunden |

Tabelle 1: Grober Zeitplan

## Ressourcenplanung

Während der Durchführung des Projektes wurden einige Hard- und Software-Ressourcen für verschiedenste Zwecke eingesetzt. Eine Übersicht aller verwendeten Ressourcen findet man im Anhang unter [A.2 Ressourcen](#_Ressourcen). Es wurde besonders darauf geachtet nur kostenfreie Software zu verwenden.

## Entwicklungsprozess

Für die Planung der Realisierung des Projektes wurde die Vorgehensweise im Vorfeld definiert. Das Projektteam entschied sich für eine Durchführung des Projektes mit dem [Spiralmodell](#_Spiralmodell) und dem Ansatz des [Test-Driven-Developments](#_Test-Driven-Development).

### Spiralmodell

Das Projektteam hat sich für ein Vorgehen mit dem Spiralmodell entschieden. Dieses ist ein iteratives Vorgehensmodell mit besonderem Augenmerk auf die Risikobetrachtung. Vor jedem Zyklus wurden die Risiken der jeweiligen Projektphase von dem Projektteam besprochen und Maßnahmen zur Vermeidung dieser Risiken festgelegt. Der iterative Ansatz ermöglicht ein agileres Vorgehen, als das Wasserfallmodell. Das Modell fördert eine fehlerfreie Programmierung und ermöglicht es durch die kurzfristigen Zyklen die Kontrolle über die zeitliche Durchführung des Projektes zu behaltens.

### Test-Driven-Development

Bei dem Test-Driven-Development ([TDD](#_Abkürzungsverzeichnis)) wurden sogenannte Komponententests (Unit-Tests) genutzt, um die Entwicklung der Software zu steuern. Dabei wurde nach folgendem Zyklus gearbeitet:

1. **Unit-Test schreiben**: Es wird ein Komponententest geschrieben, welcher die Funktionalität einer Methode überprüft. Dieser Test schlägt zunächst fehl, da noch keine Implementierung vorhanden ist.
2. **Code implementieren**:Der Code wird soweit implementiert, bis der Test nicht mehr fehlschlägt, also die gewünschte Funktionalität erreicht wurde.
3. **Code refaktorisieren**:Der funktionierende Code wird aufgeräumt. Das heißt, dass unnötige Dopplungen und überflüssiger Code entfernt werden, um die Verständlichkeit und die Performance der Methoden zu verbessern.

Diese Art der Programmierung fördert einen sauberen Programmcode und wirkt dem Auftreten von Fehlern im Code entgegen. Das Modell lässt sich mit dem Spiralmodell kombinieren, da die Tests eine Möglichkeit darstellen, um die Risiken in der Implementierungsphase zu reduzieren.

# Analysephase

Während der Analysephase wurde der Ist-Zustand ermittelt. Am Ende der Analysephase steht das [Lastenheft](#_Lastenheft_2), welches die ausformulierten Anforderungen aus Anwendersicht enthält.

## Ist-Analyse

Zu Beginn der Analysephase wurde der Ist-Zustand des Projektes ermittelt, da es für das Projekt keinen direkten Auftraggeber gab, wurden die üblichen Systeme eines Lagerbetriebes erfasst und mit Blick auf das Projekt eingeordnet. Die Systeme, welche für das Projekt relevant sind wurden in der weiteren Planung und den Entscheidungen in der Entwurfsphase berücksichtigt.

## Nicht-monetäre Vorteile

Abgesehen von der bereits in [1.3. Projektbegründung](#_Projektbegründung) erwähnten Kostenersparnis bietet das neue System noch weitere Vorteile. So wird durch die Anwendung die Verwaltung der Hardware-Komponenten für die Mitarbeiter erheblich vereinfacht, außerdem ist eine leichtere Absprache der Mitarbeiter auch über mehrere Abteilungen möglich, da diese auf demselben Datenbestand arbeiten.

## Lastenheft

Das Lastenheft wurde am Ende der Analysephase erstellt. Dieses enthält die Kriterien, die die Entwickler aus Sicht der Anwender formuliert haben. Die Anforderungen wurden der Priorität absteigend nach Muss-, Soll- und Kann-Kriterien gestaffelt. Ein Auszug des Lastenhefts findet sich im Anhang unter [A.3 Lastenheft](#_Lastenheft).

## Zwischenstand

|  |  |  |  |
| --- | --- | --- | --- |
| Projektphase | Geplante Stunden | Benötigte Stunden | Differenz |
| Ist-Analyse | 1 | 0,5 | -0,5 |
| Erstellen eines Lastenheftes | 1 | 1 | 0 |

# Entwurfsphase

Während der Entwurfsphase wurde die technische Umsetzung des Projektes geplant und das Design der Benutzeroberfläche festgelegt. Am Ende der Entwurfsphase steht das [Pflichtenheft](#_Pflichtenheft_2).

## Zielplattform

Als Zielplattform der Anwendung wurde ein Windows-System mit installiertem *.NET-Framework* gewählt. Das Aufbauen der Anwendung nach dem Schichtenmodell ermöglicht es die Datenbank modular, auf einem separaten System zu betreiben, so ist es auch möglich mehrere Client-Anwendungen auf verschiedenen Windows-Rechnern gegen dieselbe, zentral gelegene Datenbank zu betreiben. Als Datenbanksystem wurde *MariaDB* gewählt, da hier die meiste Erfahrung im Projektteam vorhanden war und diese kostenfrei zur Verfügung steht. Die verwendete Programmiersprache wurde mit Blick auf das Zielsystem (Windows) gewählt. Das Team hat sich dafür entschieden die Anwendung in der Sprache *C#* zu entwickeln, da hier der größte, gemeinsame Wissensstand existierte.

## Architekturdesign

Für die Umsetzung des Projektes wurde vom Projektteam vereinbart die einzelnen Module der Anwendung in Schichten darzustellen. Dies fördert eine modulare Programmierung und somit die Wartbarkeit und Wiederverwendbarkeit der Anwendung, außerdem erleichtert es das Testen der einzelnen Komponenten, da diese unabhängig voneinander funktionieren. Eine Übersicht über die Schichten kann man dem Anhang unter [A.4 Schichtenmodell](#_Schichtenmodell) entnehmen.

## Benutzeroberfläche

Um ein ergonomisches Design zu gestalten, wurden bereits in der Entwicklungsphase Konzepte für eine Benutzeroberfläche erstellt (*Mockups*). Diese ermöglichen eine schnellere Erstellung der Oberfläche und eine strukturierte Darstellung der verwendeten Komponenten. Alle Mockups wurden mit dem Online-Tool unter <http://www.draw.io> erstellt und können im Anhang unter [A.5 Oberflächenentwürfe](#_Oberflächenentwürfe) eingesehen werden. In dem Hauptfenster soll ein Auswahlmenü mit den Hardware-Komponenten erstellt werden. Im Zentrum des Fensters befindet sich die Tabelle, welche die Daten der Komponenten anzeigt und eine Sortierung ermöglicht.

## Datenbankentwurf

Um den Lagerbestand in der relationalen Datenbank korrekt abzubilden wurde im Vorfeld ein *Entity-Relationship-Model* ([ERM](#_Abkürzungsverzeichnis)) erstellt. Dieses findet sich im Anhang unter [A.6 ER-Modell](#_ER-Modell). Neben den Hardware-Komponenten werden auch zwei Stammdaten-Tabellen „Hersteller“ und „Schnittstelle“, sowie die Beziehung zu den Komponenten aufgezeigt. Das ER-Modell enthält außerdem Angaben über die Attribute der jeweiligen Entitäten. Das ER-Modell ermöglicht es einen Überblick über die Datenbank zu gewinnen und vereinfacht somit das Definieren von Datenbank-Regeln.

## Geschäftslogik

Durch das in Abschnitt [2.3.2 Test-Driven-Development](#_Test-Driven-Development) beschriebene Vorgehen ist es nicht möglich die Logik der meisten Komponenten vor dem Schreiben der Komponententest zu planen, daher wurde dies nur für die Teile der Anwendung übernommen, die nicht von den Tests abgedeckt wurden. Die Planung der Geschäftslogik wurde mit Hilfe von Struktogrammen übernommen, dies ermöglicht es Fehlerquellen im Vorfeld zu erkennen und zu beseitigen. Ein Beispiel für ein erstelltes Struktogramm ist im Anhang unter [A.7 Struktogramm](#_Pflichtenheft) zu finden.

## Pflichtenheft

Das Pflichtenheft wurde basierend auf den im Lastenheft spezifizierten Anforderungen und der in der Entwurfsphase getroffenen Entscheidungen erstellt. Es enthält die konkrete, technische Umsetzung der im [3.2 Lastenheft](#_Lastenheft_1) definierten Anforderungen. Ein Auszug des Pflichtenhefts ist im Anhang unter [A.8 Pflichtenheft](#_Pflichtenheft) zu finden.

## Zwischenstand

|  |  |  |  |
| --- | --- | --- | --- |
| Projektphase | Geplante Stunden | Benötigte Stunden | Differenz |
| Datenbankentwurf | 2 | 2 | 0 |
| Benutzeroberfläche entwerfen und abstimmen | 1 | 1 | 0 |
| Erstellen des Pflichtenhefts | 1 | 0,5 | -0,5 |

# Implementierungsphase

Die Implementierung des Projektes erfolgte auf Grundlage des in der Entwurfsphase erstellten Pflichtenhefts, dieses kann unter [4.5. Pflichtenheft](#_Pflichtenheft_1) nachgeschlagen werden.

## Iterationsplan

Zu Beginn der Implementierungsphase wurde ein Iterationsplan erstellt, dieser bietet eine Übersicht über die zu implementierenden Bestandteile der Software. Der Iterationsplan diente während der Durchführung als Anhaltspunkt für den Fortschritt der Implementierungsphase und entspricht dem unter [2.3.1 Spiralmodell](#_Spiralmodell) beschriebenen Verfahren. Der Iterationsplan ist im Anhang unter [A.9 Iterationsplan](#_Iterationsplan) zu finden.

## Implementierung der Datenstrukturen

Die in der Entwurfsphase definierte Struktur der Datenbank, welche unter [4.4 Datenbankentwurf](#_Datenbankentwurf) genauer beschrieben wurde, wurde mittels [SQL](#_Abkürzungsverzeichnis) in Tabellen und Attribute in eine *MariaDB*-Datenbank übersetzt. Dafür wurde zuerst eine neue Datenbank mit Hilfe der grafischen Oberfläche von *phpMyAdmin* erstellt. Anschließend wurden alle definierten Tabellen und Attribute angelegt. Die Integrität der Daten wurde mit Hilfe von Datenbank-Regeln umgesetzt.

## Implementierung der Geschäftslogik

Um die Implementierung des *C#*-Codes vorzunehmen wurde die Entwicklungsumgebung *Visual Studio Community 2015* verwendet. Zuerst wurde die Struktur des Arbeitsbereiches erstellt, dies geschah auf Basis der im Schichtenmodell ([4.2 Architekturdesign](#_Architekturdesign)) definierten Schichten. Für die Unit-Tests wurde ein separates Projekt angelegt, welches eine Referenz auf das Hauptprojekt beinhaltet, um so die dort implementierten Klassen testen zu können. Die Entitäten der Datenbank wurden als eigenständige Klassen mit den jeweiligen Attributen implementiert. Auf der Schicht der Datenbankverbindung wurden sogenannte Datenzugriffsklassen (Data Access) geschrieben, welche die Kommunikation mit der Datenbank gewährleisten, sowie eine übergeordnete Klasse mit allgemeinen Methoden zum Absetzen von Abfragen an die Datenbank. Um die Konsistenz der Daten zu sichern wurden zusätzlich zu den Datenbank-Regeln auch Validatoren implementiert. Auszüge des Quellcodes können im Anhang unter [A.10 Quellcode](#_Quellcode) nachgeschlagen werden.

## Implementierung der Benutzeroberfläche

Die grafische Benutzeroberfläche wurde auf Basis der in der Entwurfsphase erstellten *Mockups* mit Hilfe der *Windows Presentation Foundation* ([WPF](#_Abkürzungsverzeichnis))erstellt. Der Code für die grafische Oberfläche wurde im sogenannten *Extensible-Application-Markup-Language-Format* ([XAML](#_Abkürzungsverzeichnis)) geschrieben, dabei wurde auch der in *Visual Studio* integrierte Designer verwendet. Einen Screenshot der Anwendung findet sich im Anhang unter [A.11 Screenshot der Anwendung](#_Screenshot_der_Anwendung).

## Zwischenstand

|  |  |  |  |
| --- | --- | --- | --- |
| Projektphase | Geplante Stunden | Benötigte Stunden | Differenz |
| Erstellen des Datenbankschemas | 2 | 2 | 0 |
| Schreiben von Unit-Tests | 2 | 2 | 0 |
| Implementieren des C#-Codes | 7 | 7 | 0 |
| Umsetzung der Oberfläche | 2 | 3 | +1 |

# Abnahmephase

In der Abnahmephase wurde von dem Projektteam ein Oberflächentest durchgeführt, um die grafische Benutzeroberfläche unter den Gesichtspunkten der Benutzerfreundlichkeit und Ergonomie zu überprüfen. Im Zuge dieses Tests wurde auch ein Datenkonsistenztest durchgeführt, um die erstellten Datenbank-Regeln und die implementierten Validatoren auf ihre Korrektheit zu testen.

## Zwischenstand

|  |  |  |  |
| --- | --- | --- | --- |
| Projektphase | Geplante Stunden | Benötigte Stunden | Differenz |
| Abnahmetest | 1 | 1 | 0 |

# Dokumentation

Im Rahmen der Projektdurchführung wurde eine vollständige Entwicklerdokumentation geschrieben, welche die Projektphasen beschreibt, sowie einen Überblick über die verwendeten Techniken gibt. Die Dokumentation enthält außerdem alle für das Projekt relevanten Diagramme und Modelle um spätere Anpassungen und Weiterentwicklungen zu erleichtern.  
Des Weiteren wurde der *C#*-Code mit [XML](#_Abkürzungsverzeichnis)-Kommentaren versehen, welche sich problemlos bei jedem Aufruf der entsprechenden Klasse/Methode anzeigen lassen. Die Kommentare wurden für einen besseren Überblick in eine Datei exportiert.

## Zwischenstand

|  |  |  |  |
| --- | --- | --- | --- |
| Projektphase | Geplante Stunden | Benötigte Stunden | Differenz |
| Erstellen der Projektdokumentation | 4 | 4 | 0 |

# Fazit

## Soll-/Ist-Vergleich

Das Projektziel wurde erreicht und alle Muss-Kriterien wurden realisiert. Während der Durchführung des Projektes sind nur geringfügige Abweichungen von dem vorher definierten Zeitplan aufgetreten, da einige Anpassungen an der Oberfläche nur mittels direkter Manipulation der [XAML](#_Abkürzungsverzeichnis)-Dateien möglich waren. Die Abweichungen hatten jedoch keine Auswirkungen auf die geplante Gesamtlänge des Projektes.

|  |  |  |  |
| --- | --- | --- | --- |
| Projektphase | Geplante Stunden | Benötigte Stunden | Differenz |
| Analysephase | 2 | 1,5 | -0,5 |
| Entwurfsphase | 4 | 3,5 | -0,5 |
| Implementierungsphase | 13 | 14 | +1 |
| Abnahmetest | 1 | 1 | 0 |
| Dokumentation | 4 | 4 | 0 |
| Gesamt | **24** | **24** | **0** |

## Aus dem Projekt Gewonnenes

Das Projektteam konnte einige neue Erkenntnisse über Datenbankentwicklung und das Gestalten von ergonomischen Oberflächen gewinnen. Des Weiteren konnte viel Erfahrung in der Durchführung eines vollständigen Projektes gesammelt werden, dabei bemerkte das Projektteam, dass es besonders wichtig ist eine umfassende Planung zu erstellen und eventuelle Risiken bereits früh in der Projektplanung zu berücksichtigen.

## Ausblick

Dank der modularen Programmierung, welche auch durch den Aufbau des Projektes in Schichten und dem objektorientierten Ansatz realisiert wurde ist es möglich die Anwendung zu erweitern. In Zukunft könnten weitere Entitäten erfasst werden oder einige der im Lastenheft unter [3.2 Lastenheft](#_Lastenheft_1) definierten Kann-Kriterien realisiert werden. Dies würde den Nutzen der Anwendung weiter steigern und somit die Arbeit mit dem System verbessern.

# Anhang

## Detaillierter Zeitplan

|  |  |  |  |
| --- | --- | --- | --- |
| Phasen | Stunden | | |
| Analysephase |  |  | **2** |
| 1. Ist-Analyse |  | 1 |  |
| 1. Erstellen des Lastenhefts |  | 1 |  |
| Entwurfsphase |  |  | **4** |
| 1. Datenbankentwurf |  | 2 |  |
| * 1. Festlegen der Entitäten | 1 |  |  |
| * 1. Datenbank-Modellierung | 1 |  |  |
| 1. Benutzeroberfläche entwerfen und abstimmen |  | 1 |  |
| 1. Erstellen des Pflichtenhefts |  | 1 |  |
| Implementierungsphase |  |  | **13** |
| 1. Erstellen des Datenbankschemas |  | 2 |  |
| 1. Schreiben von Unit-Tests |  | 2 |  |
| 1. Implementieren des C#-Codes |  | 7 |  |
| 3.1 Implementieren der Entitäts-Klassen | 1 |  |  |
| 3.2 Implementieren der Datenzugriffs-Klassen | 2 |  |  |
| 3.3 Implementieren der Validatoren | 1 |  |  |
| 3.4 Implementieren der Geschäftslogik | 2 |  |  |
| 1. Umsetzung der Oberfläche |  | 2 |  |
| Abnahmetest |  |  | **1** |
| 1. Abnahmetest durch das Projektteam |  | 1 |  |
| Dokumentation |  |  | **4** |
| 1. Erstellen der Projektdokumentation |  | 4 |  |
| Gesamt |  |  | **24** |

## Ressourcen

**Hardware:**

* Arbeitsplatz
* Rechner
* Bildschirm

**Software:**

* Visual Studio Community 2015 (Entwicklungsumgebung)
* Windows 7 Professional (Betriebssystem)
* GitHub (Versionsverwaltung)
* GitHub-Plugin für Visual Studio
* draw.io (Tool für Mockups und ER-Modelle)
* Windows Presentation Foundation (Framework für [GUIs](#_Abkürzungsverzeichnis) unter C#)
* Notepad++ (Editor)
* .NET MySQL-Treiber (Treiber für die Datenbankverbindung)
* Struktogrammeditor 1.7 (Programm zum Erstellen von Struktogrammen)
* Microsoft Word und Power Point 2010 (Software für die Dokumentation und Präsentation)

## Lastenheft

**Muss-Kriterien:**

* Dauerhaftes Speichern von Komponenten mit allen definierten Merkmalen (Arbeitsspeicher, Festplatte, Grafikkarte, Hauptplatine, Monitor, Prozessor)
* Anzeigen von Komponenten in einer Liste
* Bearbeiten von bereits gespeicherten Komponenten
* Löschen von gespeicherten Komponenten
* Grafische Oberfläche

**Soll-Kriterien:**

* Ansprechende grafische Oberfläche
* Umrechnung in andere Einheiten (z.B. Byte -> GB)

**Kann-Kriterien:**

* Konfigurieren der Datenbankverbindung über die Oberfläche
* Löschen von mehreren Komponenten gleichzeitig
* Exportieren von Tabellen als CSV-Datei
* Importieren von Tabellen aus einer CSV-Datei

## Schichtenmodell



Abbildung 1 - Schichtenmodell

## Oberflächenentwürfe



Abbildung 2 - Startfenster (Mockup)



Abbildung 3 - Erstellen einer Komponente (Mockup)

## ER-Modell

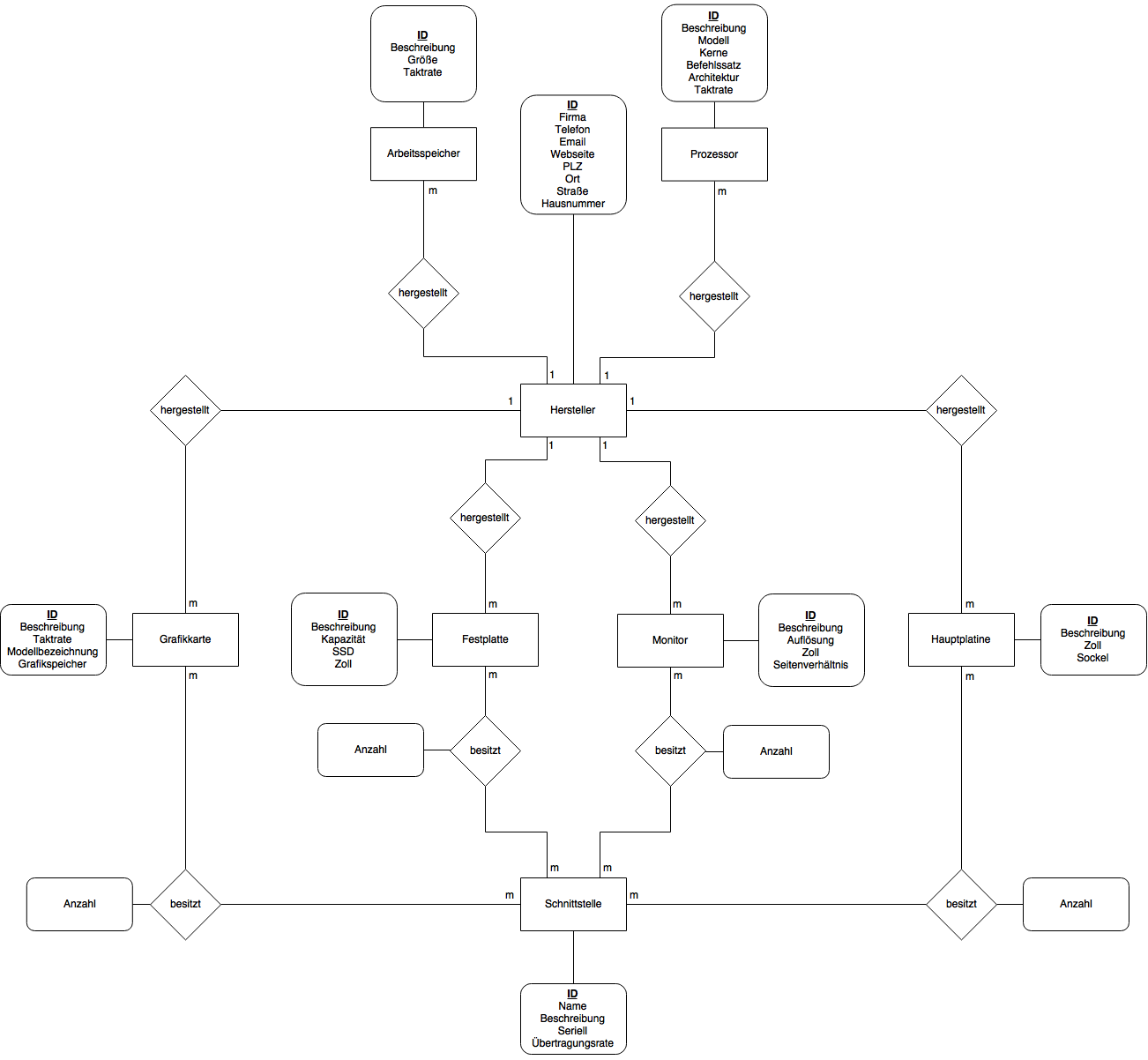


Abbildung 4 - Entity-Relationship-Model

## Struktogramm

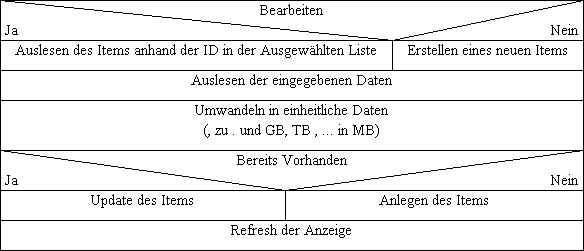


Abbildung 5 - Struktogramm: Speichern von Entitäten

## Pflichtenheft

**Muss-Kriterien:**

* Speichern vom Komponenten in einer relationalen Datenbank (MariaDB)
* Datenbankverbindung mittels MySQL-Treiber für .NET (C#)
* Selektieren von Datensätzen aus der Datenbank mittels [SQL](#_Abkürzungsverzeichnis)
* Anzeigen von Datensätzen in einer Tabelle auf der [GUI](#_Abkürzungsverzeichnis)
* Löschen von gespeicherten Komponenten aus der Datenbank mittels [SQL](#_Abkürzungsverzeichnis)
* Gestalten der grafischen Oberfläche ([GUI](#_Abkürzungsverzeichnis)) mit dem Windows Presentation Foundation Framework ([WPF](#_Abkürzungsverzeichnis))
* Die Konsistenz der Daten wird durch Validatoren gewährleistet

**Soll-Kriterien:**

* Entwicklung einer ergonomischen [GUI](#_Abkürzungsverzeichnis) mittels Mockups
* Schreiben eines Algorithmus für die Umrechnung von Einheiten in C#

**Kann-Kriterien:**

* Konfigurieren der Datenbankverbindung über ein Konfigurationsfenster in C# mittels [WPF](#_Abkürzungsverzeichnis)
* Selektieren von mehreren Einträgen mittels einer DataGrid-Tabelle auf der [WPF](#_Abkürzungsverzeichnis)-Oberfläche
* Exportieren von Tabelle als [CSV](#_Abkürzungsverzeichnis)-Datei mittels File Stream

## Iterationsplan

1. Erstellen des [IMS](#_Abkürzungsverzeichnis)-Projektes
   * Anlegen eines Visual Studio Projektes
   * Erstellen eines Repositories auf GitHub
   * Anlegen einer Verzeichnisstruktur
2. Anlegen einer relationalen Datenbank
   * Anlegen von Tabellen
   * Anlegen von Attributen und Festlegen von Datentypen
   * Erstellen von Datenbank-Regeln
3. Implementieren von Entitätsklassen in C#
   * Implementieren von Klassen für jede Entität
4. Implementieren des Datenbankzugriffes in C#
   * Schreiben von Unit-Tests für die Verbindung mit der Datenbank
   * Implementieren einer allgemeinen Datenbank-Klasse
   * Implementieren von Datenbank-Klassen für jede Entität
5. Implementieren der Validatoren in C#
   * Festlegen von Validierungsregeln
   * Schreiben von Unit-Tests für die Validierung
   * Implementieren von Validatoren für jede Entität
6. Implementieren der Benutzeroberfläche in C# und [XAML](#_Abkürzungsverzeichnis)
   * Erstellen von [WPF](#_Abkürzungsverzeichnis)-Fenstern
   * Implementieren der Hintergrundlogik

## Quellcode

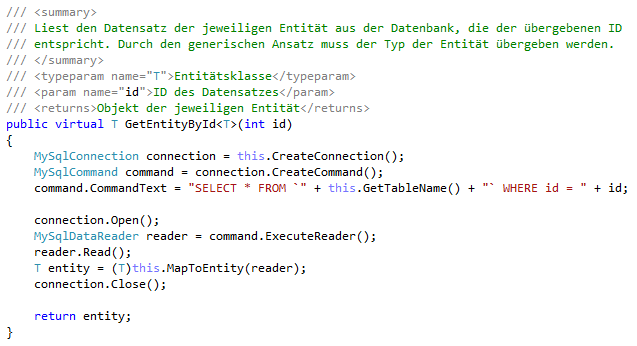


Abbildung 6 - Quellcode-Beispiel: Lesen eines Datensatzes

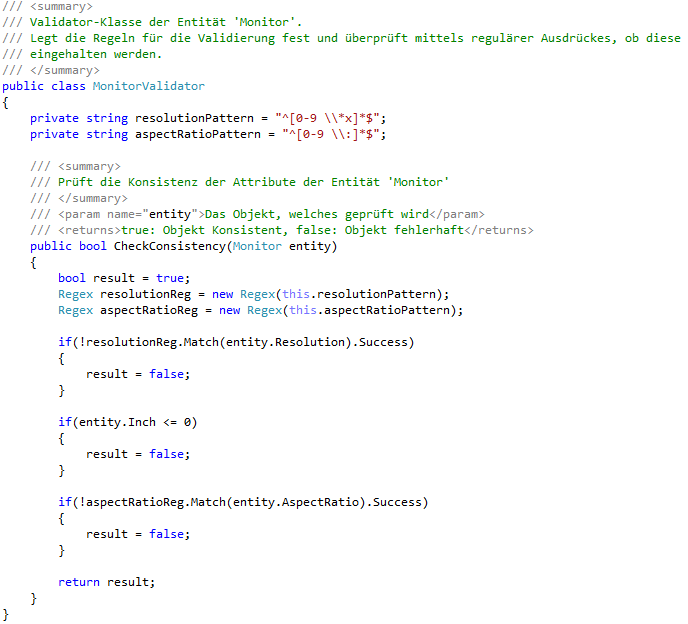


Abbildung 7 - Quellcode-Beispiel: Validator für Monitor

## Screenshot der Anwendung

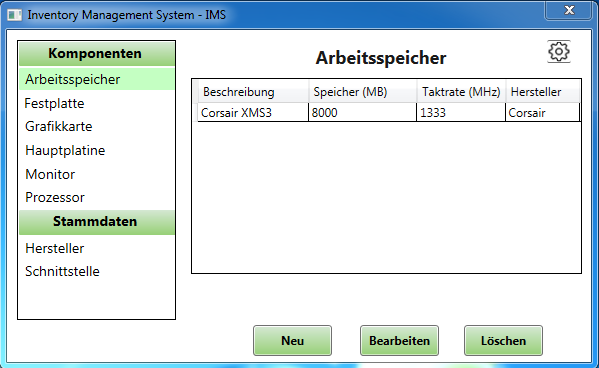


Abbildung 8 – Startfenster (Screenshot)

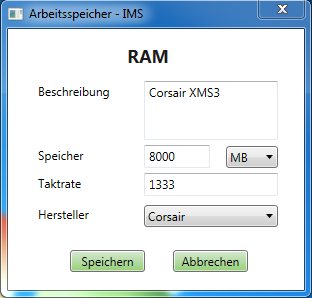


Abbildung 9 - Erstellen einer Komponente (Screenshot)