Abschlussprüfung Sommer 2023

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Netzwerkmonitoring– und Auswertetool

Desktopanwendung: Netzwerkmonitoring und Auswertetool zur Optimierung der Netzwerkqualität und Verringerung möglicher Angriffspunkte durch nicht autorisierte Devices im Netzwerk

Abgabe am 19.05.2023 an die IHK Niederbayern zu Passau

Prüfungsbewerber:
Bernecker Thomas
Schäfflerstraße 2
94315 Straubing

Ausbildungsbetrieb:
Vitesco Technologie Group AG
Siemenstraße 12
93055 Regensburg

Eidesstattliche Versicherung

Ich, Thomas Bernecker - als Autor - versichere hiermit, dass ich meine Dokumentation zur betrieblichen Projektarbeit mit dem Thema

"Netzwerkmonitoring– und Auswertetool, Desktopanwendung: Netzwerkmonitoring– und Auswertetool zur Optimierung der Netzwerkqualität und Verringerung möglicher Angriffs-punkte durch nicht autorisierte Devices im Netzwerk"

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Straubing, den 19.05.2023

I Inhaltsverzeichnis

II. Textteil / Projektbeschreibung

1. Einleitung

1.1 Projekt-Umfeld

Die Vitesco Technologies Group AG Regensburg (folgend „Vitesco“ genannt) war ein Unternehmenszweig der Continental AG. Vitesco hat als börsennotierter Automobilzulieferer über 50 Standorte weltweit und hat mit mehreren Tausend Mitarbeitern ihren Hauptsitz in Regensburg. Der Fokus des Unternehmens liegt auf Antriebstechnologien, hierbei entwickelt es Systemlösungen und Komponenten für Verbrennungs-, Hybrid- und Elektroantriebe.
Der Autor absolvierte sein Projekt in der Abteilung „Local IT“ der Vitesco Technologies Group AG in Regensburg.

1.2 Projekt-Ziel

Ziel des Projektes ist die automatische Erkennung von Netzwerk-Komponenten sowie die dynamische Zuordnung und Speicherung der Informationen in einer Datenbank. Dies alles vor dem Hintergrund von Risikominimierung, Zeitersparnis sowie die Sammlung von Daten für ein zukünftig zu entwickelndem System zur Durchführung von „Predictive Maintenance“, also eine auf selbstlernenden Modellen basierenden Früherkennung von Problemen und Einleitung von automatisierten Maßnahmen. Für Letzteres soll durch dieses Projekt die entsprechende Datenbasis gelegt werden.

1.3 Projekt-Begründung

Mit der Umsetzung dieses Projektes wird eine enorme Zeitersparnis in den technischen Fachabteilungen erwartet. Details hierzu sind der Nutzwert-Analyse zu entnehmen.
Neue Netzwerkkomponenten können schneller eingepflegt werden, was eine Zeitersparnis für die anderen Abteilungen bedeutet. Außerdem können auftretende Fehler schneller bearbeitet werden. Da es derzeit keine vergleichbare Software auf dem Markt gibt, die den infrastrukturellen Rahmen der Vitesco erfüllt, ist von vornherein die Vorgabe, eine individuell gefertigte Software zu erstellen. Dies auch vor dem Hintergrund, dass auf diesem Weg jederzeit Erweiterungen durchgeführt werden können.

1.4 Projekt-Schnittstellen

Die Anwendung verarbeitet zwei CSV- Dateien und eine weitere Texdatei die aus dem Internet von der Webseite der IEEE gedownloadet werden. Es wird eine SQLite Datenbank verwendet, da diese eine hohe Portabilität des Programmes ermöglicht und diese Technologie bei Vitesco bereits bewährt im Einsatz ist. Die Anwender sind die Mitarbeiter der Abteilung “Local IT“.

Der Leiter der Abteilung „Local IT“, Herr Hagg ist der interne Auftraggeber. Weitere Stakeholder sind die Mitarbeiter der Abteilung „Local IT“, die durch die neue Anwendung entlastet werden sollen.

2. Projektplanung

2.1 Projektphasen und Zeitplanung

Für die Umsetzung des Projektes wird ein lineares Phasenmodell gewählt, da die einzelnen Projektphasen sauber voneinander getrennt werden können und auf diese Art eine solide Planbarkeit des Projektverlaufes gegeben ist. Folgende Phasen werden hierbei eingeplant:

Projektphase	Geplante Zeit
Analyse	5 Std.
Entwurf	10 Std.
Implementierung	52 Std.
Abnahme	3 Std.
Einführung	3 Std.
Dokumentation	7 Std.
Gesamt	**80 Std.**

TABELLE 1: GROBE ZEITPLANUNG

Die Umsetzung erfolgt im Rahmen der bei Vitesco für Praktikanten üblichen 35-Stunden-Woche. Das Projekt ist angelegt auf drei Wochen Durchführungszeitraum im Rahmen des Praktikums.

2.2 Abweichungen vom Projektantrag

Abweichend zum Projektantrag wurde für die Analyse-Phase eine Stunde mehr Arbeitszeit veranschlagt und diese dem geplanten Puffer für die Implementierungs-Phase entnommen.

2.3 Ressourcenplanung

Für die Umsetzung werden die folgenden Ressourcen sowie Technologien eingeplant:

- Büroarbeitsplatz
- Laptop mit Administratorenrechten
- Betriebssystem: Windows 10
- IDE: Microsoft Visual Studio 2022
- Versionierung: Git
- Datenbank: SQLite Framework (SQLite.Core)
- Json Framework: Newtonsoft.Json

- Micro-ORM Framework: Dapper
- Entwickler für Review des Codes
- Mitarbeiter zum Usability-Testing

Ebenso ist der Einsatz eines erfahrenen Senior-Entwicklers für die Durchführung der Code-Reviews eingeplant, um die Korrektheit und die Qualität des Codes sicherzustellen.

2.4 Hinweis zum Entwicklungsprozess

Die Anwendung wird in der Implementierungs-Phase agil entwickelt, um Teile der Software bereits während des Entwicklungsprozesses funktionsfähig zu übergeben, damit eine parallele Überprüfung sowie die Durchführung von Tests ermöglicht werden können. Insofern folgt der grundlegende Entwicklungsprozess einem linearen Phasenmodell, wird aber in der eigentlichen Implementierung agil ohne die Anwendung von Scrum o.ä. durchgeführt („Hybrides Vorgehensmodell“).

3. Analysephase

3.1 IST-Analyse

Die IST-Analyse stellt die Grundlage zur weiteren Planung der Projekt-Implementierung dar und ist unverzichtbar für eine solide Planung des Vorgehens.

3.1.1 Grundlagen zur IST-Analyse

Es wurden zur IST-Analyse verschiedene Quellen herangezogen: ServiceNow (Asset-Management), QIP (IP-Adresszuweisung), Log-Dateien aus den Switches und die Herstellerinformationen. Da die Log-Datei rollierend überschrieben wurde, kann diese nur einen Zeitraum von 4 bis 6 Stunden abdecken und nur punktuell angefragt werden.
Diese punktuelle Analyse muss zeitlich erweitert werden, auch das Heranziehen der anderen Informationen kann automatisiert werden und somit die Analyse beschleunigt und erweitert werden.

3.1.2 Kostenplanung bei eigener Entwicklung

Die Kosten der Umsetzung des Projektes werden im ersten Schritt kalkuliert. Neben Personalkosten, fallen auch Ressourcenkosten an. Diese beinhaltenen die in der Ressourcenplanung bereits erfassten Ressourcen und werden in einen kalkulatorischen Satz, der durch die Buchhaltungs-Abteilung vorgegeben wurde, verrechnet. Für die Kalkulation der Personalkosten wurde der Stundensatz für Auszubildende von 10,00 Euro verwendet, da in diesem Fall keine Informationen über die Fachabteilungen freigegeben wurden. Auch die Mitarbeiter werden mit einer Pauschale von 25,00 Euro berechnet. Für die Ressourcenkosten wurden 40 Cent pro Stunde Strom veranschlagt und weitere Kosten betragen 30 Cent je Stunde.

Personalkosten für den Autor:

10,00 Euro / Stunde * 80 = 800,00 Euro

Personalkosten für das Fachpersonal:

25,00 Euro * 6 Stunden = 150,00 Euro

Ressourcenkosten:

0,40 Euro * 80 Stunden = 32,00 Euro (für Strom)

0,30 Euro * 80 Stunden = 24,00Euro (für sonstige Kosten)

Gesamtkosten:

800,00 Euro (Stundenlohn)

+ 150,00 Euro (Fachpersonal)

+ 32,00 Euro (Stromkosten)

+ 24,00 Euro (sonstige Ressourcenkosten)

= Projektkosten: 1.006,00 Euro

3.1.3 Amortisationsrechnung

Aufgrund der hohen Anzahl der bisherigen Tickets und der Bearbeitungsdauer wurde analysiert, dass eine einzelne Bearbeitung ca. ein bis zwei Werktage benötigt. Ein Ausfall oder eine Wartezeit für eine andere Abteilungen wurde nicht mit einbezogen, da diese zu ungenau ausgefallen wären. Wenn zum Beispiel eine Abteilung auf die Einpflege eines Sensors in das System wartet, steigen die Kosten rapide, da ggf. ein Produktionsausfall verursacht wird.
Anfragen fallen im Durchschnitt laut der bisherigen Statistiken etwa 2 mal pro Woche an.

Entsprechend wurde die folgende Amortisations-Rechnung aufgestellt:

Position	Berechnung
Personalkosten pro Stunde	25,00 Euro (Inklusive Ressourcen-Kosten)
Projektkosten	1.006,00 Euro
Ersparnis pro Tag:	2 Tage * 25,00 Euro * 8 Stunden = 400,00 Euro
Ersparnis pro Jahr	104 Anfragen * 400,00 Euro = 41.600 Euro

TABELLE 2: AMORTISATIONS-RECHNUNG

Hieraus ergibt sich, dass die Amortisation (Break-Even-Point) bereits spätestens nach der dritten Anfrage eintritt. Die Wirtschaftlichkeit der Eigenentwicklung ist also klar erkennbar.

3.2 Nutzwertanalyse

Die Nutzwertanalyse wurde erstellt, um neben der Amortisations-Analyse zu ermitteln, inwieweit sich zusätzlich zum finanziellen Aspekt weitere Faktoren einer entsprechenden Entwicklung auswirken.

Einfluss-Faktor	Gewichtung	Vorher	Nachher
Kosteneinsparung	30%	0	10
Mitarbeiter-Zufriedenheit	20%	3	8
Effizienz der Auswertungen	20%	2	7
Zukunftssicherheit des Systems	20%	0	7
Verfügbarkeit des Systems	10%	0	6
Summe		**5**	**38**

TABELLE 3: NUTZWERTANALYSE

Hier ist erneut ersichtlich, dass auch die „weichen Einflussfaktoren“ eine deutliche Verbesserung der Gesamt-Situation wohl erreichen werden.

3.3 Anwendungsfälle

In der Vergangenheit trat immer wieder das Problem auf, dass Informationen über unterschiedliche Geräte im Netzwerk gebraucht wurden. Sei es wegen verdächtigen Verhaltens oder auch einfach um simple Auswertungen (z.B. Nutzungszeiten, Belastungen, etc.) zu erstellen. Diese waren jedoch nicht jedem IT-Mitarbeiter leicht zugänglich. Umständliche Anfragen an das jeweilige Netzwerkteam mussten gestellt werden, welche oftmals wegen eines hohen Arbeitsaufkommens auch erst nach einigen Tagen bearbeitet werden konnten.
Benötigte Informationen waren hier überwiegend nur allgemeine Daten zu den Netzwerkgeräten wie MAC-Adresse, Hostname, IP-Adresse oder verwendeter Access-Point.
Beispielsweise könnte folgende Situation im täglichen Geschäft auftreten: Es wurde von einem Mitarbeiter aus der Entwicklungsabteilung gemeldet, dass eines seiner Messgeräte die Verbindung zum internen Netzwerk verloren hat. Aufgrund der eingeschränkten möglichen Bedienung des Sondergerätes kann nur der Hostname an die IT-Abteilung weitergegeben werden. Der IT-Betreuer steht nun vor der Problematik, dass er, um weitere Daten des Gerätes zu erhalten und mögliche Gründe für die Ursache des Problems zu finden, entweder selbst physisch an das Gerät herantreten oder sich per Anfrage an das Netzwerkteam wenden muss.

3.4 Qualitätssicherung

Die Änderbarkeit und damit Skalierbarkeit des neuen Systems wird erreicht durch die Abkapselung zwischen Datenbank und GUI. Für die Effizienz wurden in jeder Funktion Zeitmessungen vorgenommen, um die Performance zu verbessern.

Zur Sicherstellung einer konsequenten Qualitätssicherung des Entwicklungs-Prozesses, wurden Usability-Tests durch die Mitarbeiter durchgeführt. Dies stellte auch sicher, dass die Akzeptanz des neuen Systems von vornherein gewährleistet war.
Eine Funktion zur Prüfung der übertragenen Daten auf Vollständigkeit wurde entwickelt, implementiert und getestet.
Durch die Verwendung von Git wurde die Wiederherstellbarkeit vorheriger Versionen gewährleistet sowie dem Umstand, Branches anzulegen, Vorschub geleistet.
Die Funktionalität wurde durch einen kontinuierlichen Test-Prozess im Rahmen des agilen Umsetzungsprozesses während der Implementierungs-Phase überprüft. Auf die separate Integration eines Test-Driven-Developments wurde auf Grund des überschaubaren Projekt-Umfanges verzichtet.

3.5 Lastenheft/Fachkonzept

Im Vorfeld wurde durch den Leiter der Local-IT in Zusammenarbeit mit dem Autor ein Lastenheft erstellt, um den Gesamtrahmen des Projektes abzustecken. Auszüge aus dem Lastenheft/Fachkonzept, befinden sich in der Anlage. Auf die Erstellung eines Pflichtenheftes wurde in Rücksprache mit dem Leiter Local IT verzichtet, da im Rahmen der Implementierungs-Phase ein agiles Vorgehenskonzept angewandt wurde.

4. Entwurfsphase

4.1 Zielplattform

Die Zielplattform wurde durch die betrieblichen Rahmenbedingungen vorgegeben: Die Anwendung soll auf einen Windows Server 2019 mit einen .NET 4.8 Framework laufen und es muss eine SQLite Datenbank verwendet werden. Auch die Verwendung von C# wurde vorgegeben. An dieser Stelle bestand keinerlei Einflussmöglichkeit durch den Autor, da die zukünftige Weiterentwicklung durch den gewählten Technologie-Stack auch von anderen Entwicklern der Vitesco vorgenommen werden kann.

4.2 Architektur-Design

Wie bereits vorgetragen werden Datenbank und GUI voneinander getrennt. Hierfür wurde die Variante des „Prototypings“ gewählt, um Schritt für Schritt neue Funktionalitäten für die Mitarbeiter und damit Tester sowie das Code-Review zur Verfügung zu stellen.

4.3 Auswahl des ORM-Frameworks

Im Rahmen der technologischen Vorgaben durch die Abteilung „Local IT“ wurde das ORM-Framework nicht vorgegeben. Deshalb musste hier eine Entscheidung auf Basis von Funktions-Recherchen getroffen werden. Die Entscheidungs-Matrix beinhaltet alle in Frage kommenden ORM-Frameworks:

Kriterium	Dapper	Entity	Hibernate
Leistung	3	2	1
Einfachheit in der Anwendung	3	1	0
Datenbankunterstützung	3	2	2
Dokumentation	3	2	2
Support durch eine Community	2	2	2
Gesamt:	**14**	**9**	**7**

TABELLE 4: AUSWAHL-MATRIX FÜR DAS ORM-FRAMEWORK

Auf Basis der vorstehenden Entscheidungsmatrix wurde das Framework Dapper gewählt. Diese Entscheidung wurde durch den Leiter der Abteilung Local IT freigegeben.

4.4 Entwurf der Benutzeroberfläche

Durch C# WPF war eine schnelle und einfache Umsetzung einer GUI möglich. Einfaches „Drag and Drop“ verringerte den Zeitaufwand in der Entwicklungs- und Implementierungsphase.
Ein Mock-Up wurde direkt beim Erstellen des Lastenheftes mit angefertigt, um ein besseres Verständnis zu erhalten und den betroffenen Mitarbeitern direkt eine grafische Darstellung des zu erwartenden Systems zu ermöglichen.
Die im Corporate Design angegebenen Farben und Positionierungen wurden verwendet, ebenso das Firmenlogo. Dies war zwar keine Vorgabe laut Lastenheft, entspricht aber der betriebsüblichen Praxis.

Beispielentwürfe der Oberfläche befinden sich im Anhang.

4.5 Datenbank-Modell

Das Datenbank-Modell gestaltet sich sehr einfach, da nur eine einzige Tabelle benötigt wurde. Dennoch wurden bereits im vorherein die notwendigen Grundlagen der Normalisierung berücksichtigt, um ggf. Erweiterungen zu implementieren, z.B. wenn zukünftig eine Gruppierung in Geräte-Kategorien ermöglicht werden soll.

Das Datenbank-Modell befindet sich im Anhang.

4.6 Geschäftslogik

Die Geschäftslogik hat der Autor im Rahmen des agilen Entwicklungs-Prozesses nach und nach erstellt. Zunächst wurden die Daten aufgeteilt auf die einzelnen Clients und dann zusammengeführt. Danach wurde der Hersteller aus der gedownloadeten Datei anhand der MAC-Adresse herausgesucht und hinzugefügt. Anschließend wurde eine Liste der Objekte an die

Datenbank über den Micro-ORM übergeben. Die GUI enthält die Tabelle aus der Datenbank und wird zum Schluss mit einer Suchfunktion ausgestattet.

Die Anwendung wird auf einem Server bereitgestellt, womit die IT-Mitarbeiter Zugriff haben.

4.7 Umsetzung der Maßnahmen zur Qualitätssicherung

Es wurden kontinuierlich Anwendertests (Black-Box-Verfahren) durchgeführt und kritische Funktionen mit Zeitmessung versehen. Über das gesamte Projekt hin werden Log-Dateien erstellt. Die Änderbarkeit wurde erreicht durch die Abkapselung der Daten von der GUI. Eine Funktion zur Prüfung der übertragenen Daten auf Vollständigkeit wurde entwickelt, implementiert und getestet sowie kontinuierlich angewandt. Ebenso wurde die durch Git angedachte Möglichkeit zur Durchführung von Code-Reviews durch den zugeteilten Senior-Developer umgesetzt. Eventuelle Anmerkungen konnten auf diese Art direkt umgesetzt werden und dann durch den Senior-Developer committet, also in den Quellcode hinein bestätigt werden.

5. Implementierungs-Phase

5.1 Implementierung der Datenbank-Strukturen

Die Datenbank wird, sofern sie noch nicht vorhanden ist, automatisch generiert. Es werden hierfür die Attribute einer Klasse dem Micro-ORM übergeben und mit einen SQL-Create-Commando erstellt. Über diese Funktion können bei zukünftigen Updates auch ggf. ALTER-TABLE-Statements durchgeführt werden, um die Datenbank-Struktur auf neue Funktionen hin anzupassen.

5.2 Implementierung der Benutzeroberfläche

Die Benutzeroberfläche wurde in WPF erstellt anhand des Mockups. Diese wurde so angelegt, dass Bestandteile der Oberfläche ausgekoppelt bzw. ersetzt werden können. Es wurde im Hauptfenster eine Navigationsleiste angelegt. Durch die Navigationsleiste werden im Hauptfenster die jeweiligen Unterfenster aufgerufen und angezeigt. Es wurden die Farben verwendet, die das Corporate Design vorsieht.

5.3 Implementierung der Geschäftslogik

Zunächst wurden die Klassen, Funktionen und Modelle angelegt.

5.3.1 Klasse „NetwerkclientsModel()“

Die erste Klasse war NetwerkclientsModel. Dies ist die Klasse, die zum Auffüllen der Daten aus den verschiedenen Quellen dient. Nun wurde die Logik-Klasse erstellt, dort wurde die Funktion loadMacTable() erstellt, die die Mag-Log.csv einliest und auf einzelne Informationen zerlegt sowie einen Model hinzugefügt, so dass jedes Model-Objekt eine Mac-Adresse und die

anderen dazugehörigen Informationen bekommen hat. Dieses Model wurde zu einer Liste hinzugefügt, solange die Maclog-Datei komplett abgearbeitet wurde.

5.3.2 Funktion „smelting()“

Die fertige Liste wurde, dann der smelting() Funktion übergeben. Diese Funktion prüft jede Macadresse in der QIP-Export.csv. Wurde die MAC-Adresse gefunden, wird dem Model die vorhandene Information hinzugefügt. Sofern die MAC-Adresse nicht gefunden wird, wird eine entsprechende Warnung geschrieben.

5.3.3 Funktion „producerListConvert()“

Die Funktion producerListCovert() lädt mit den .Net-Webclient eine Hersteller-Liste von der Webseite der IEEE herunter. Danach wird von jedem Model die MAC-Adresse so zurechtgeschnitten, dass nur noch die ersten 6 Stellen als String vorhanden sind und damit eine Aufschlüsselung auf den Hersteller möglich ist. Dieser String wurde in der Text-Datei (Herstellerliste) gesucht. Dort wo es einen Treffer gab, wurden die Hersteller herausschnitten und dem Model hinzugefügt. Falls es keinen Treffer gab oder ein Doppel-Eintrag gefunden, wurde, wurde dem Model unter seinem Hersteller eine passende Notiz hinzugefügt.

5.3.4 Klasse „ClientsRepository()“

Nun erstellte der Autor die Klasse „ClientsRepository“, die als Schnittstelle für die SQLite-Datenbank dient. Dafür hat der Autor zuerst die testExsi() Funktion geschrieben, die eine Prüfung vornimmt, ob die Datenbank vorhanden ist und falls nicht, die Datenbank generiert. Dach erstellte der Autor die Funktion testeCon(). Diese prüft ob zu dieser Datenbank eine Verbindung aufgebaut werden kann.

5.3.5 Prüfen der Tabelle

Nun folgt die checkTable() Funktion, die prüft ob die Datenbank, die entsprechende Tabelle beinhaltet. Falls dies nicht der Fall sein sollte, wird mit Hilfe des Framework Dapper und der Liste an Attributen in der Klasse „NetwerkclientsModel“ die Tabelle angelegt. Als nächsten Schritt hat der Autor die Standard-Funktionen vorbereitet: INSERT, UPDATE und DELETE. Zurück in der Logic-Klasse hat der Autor die fillDb() Funktion erstellt die die gesamte Liste an Objekten über das Framework in die Datenbank übergibt . Der Autor hat in dieser Funktion nun die anderen Funktionen zum Komplettieren der Liste aufgerufen und dann jedes NetwerkclientsModel mit dem QUERY-Befehl des Dapper-Mirco-ORM weitergegeben.

5.3.6 Funktion „isDbOk()“

Als nächsten Schritt hat sich der Autor entschieden, die Prüfung auf Vollständigkeit, durch die Funktion isDbOk(), der übertragenen Daten zu gewährleisten. Diese Funktion zählt jede MAC-Adresse in der der Ausgangsdatei (Maglog.csv). Danach wurden anhand des Zeitstempels in der Datenbank die Einträge gezählt. Sollte ein Unterschied festgestellt werden, wurde eine

Löschung der in der Datenbank befindlichen Daten programmiert anhand des Zeitstempel. Somit ist sichergestellt, dass durch das Programm keine Doppeleinträge entstehen. Durch eine Schleife wurden diese Vorgänge wiederholt, bis die Daten vollständig komplettiert wurden.

5.3.7 Umsetzung der GUI

Nun entstand die GUI, durch einfaches „Drag and Drop“, anhand der Mockups. Im Programmcode der GUI wurde nun ein Objekt der Logic-Klasse initialisiert und die Funktionen für die entsprechenden Anwendungen bereitgestellt. Über das Textfeld, das zum Suchen verwendet wird, wird der zu suchende Text sowie die eingestellten Parameter des Dropdown-Menüs an die Logic-Klasse übergeben, die wiederum die ClientsRepository verwendet, um die Daten in der Datenbank zu suchen und zurückzugeben. Somit wurde erreicht, dass die GUI leicht austauschbar ist. Die durch die Suche erhaltenen Daten werden in einem GridView angezeigt. Nun wurde die Suche erweitert und mit einem Dropdownmenu und einem Radio-Button verknüpft. Das Dropdownmenu beinhaltet die Spaltennamen der Tabelle aus der Datenbank und wird als Suchparameter verwendet. Die Radio-Buttons steuern die Begrenzung zwischen den gesamten Informationen und des jeweiligen Tages. Letzteres wurde aus Performance-Gründen standardmäßig gesetzt.

5.3.8 Entwicklung der Einstellungen

Als nächstes wurden nun die Einstellungen entwickelt. Durch einfache Labels und Dialog-Boxen wurden die verschiedenen Einstellungen in die Datei Settings.json ausgelagert um hart programmierte Konfigurations-Stellen ersetzen können. Dadurch ist auch die Klasse Settings entstanden, die zwei Funktionen hat. Diese liest aus der Json-Datei und schreibt die entsprechenden Werte auch dorthin zurück. Zum Abschluss wurde noch ein Warnungen-Fenster implementiert, das Logikklasse des Autors verwendet um doppelte Mac-Adressen zum gleichen Zeitpunkt anzuzeigen in einem Gridview. Danach wurde auch hier eine Filterung zwischen „Heute“ und „Gesamt“ hinzugefügt.

5.3.9 Parallele Durchführung von Tests

Zur selben Zeit als der Autor mit dem Erstellen des GUI beschäftig war, war ein Fach-Mitarbeiter bereits damit beschäftigt, die Datenbank auf Fehler zu überprüfen (White-Box-Test). Parallel dazu erfolgten die User-Tests, im Rahmen derer die Mitarbeiter die ersten Suchen durchführten (Black-Box-Test). Dabei wurden kleine Fehler gefunden, die direkt im Anschluss behoben wurden.

5.3.10 Quellcodes

Die entsprechenden Quellcodes befinden sich in der Anlage.

6. Abnahme-Phase

Die Abnahme wurde durch den Mitarbeiter durchgeführt, der das Projekt begleitet hat und die Tests unterstützt durch die Code-Reviews desselben Mitarbeiters. Durch das agile Vorgehen in der Umsetzungs-Phase konnten hier keine Probleme festgestellt werden und die Abnahme wurde ohne Beanstandungen erteilt. Hierdurch konnte nun die Einführung in den produktiven Betrieb eingeleitet werden.

7. Einführungs-Phase

Die Anwendung wurde auf dem Server installiert und durch die Bereitstellung der entsprechenden Verlinkung an die damit arbeitenden Mitarbeiter distribuiert. Durch den Umstand, dass die Mitarbeiter bereits während der Implementierungs-Phase mit dem Programm gearbeitet haben und während dieser Phase bereits alle Funktionen erklärt wurden, gab es keinerlei Rückfragen zur Benutzung des Programmes. Hierdurch wurde die Durchführung einer separaten Schulung vermieden, was den Vorteil der Wahl eines agilen Vorgehens in der Implementierungs-Phase, flankiert durch den Planungs-Vorteil eines linearen Vorgehensmodells, bestätigt.

8. Dokumentation

8.1 Entwickler-Dokumentation

Die Entwickler-Dokumentation wurde bereits begleitend zur Implementierungs-Phase auf Basis von „Doxygen“ erstellt und im Nachgang sowohl als HTML als auch als PDF im Programm-Entwicklungsordner bereitgestellt. Die Nutzung von Doxygen ermöglicht es, die Dokumentation auch bei zukünftigen Anpassungen jederzeit auf schnellste Art und Weise zu aktualisieren und bereit zu stellen.

8.2 Anwender-Dokumentation

Anschließend wurde noch eine kurze Anwender-Dokumentation erstellt. Da die Benutzung des Programmes sehr intuitiv ist, konnte diese auf ein Mindestmaß reduziert werden.
Ein Auszug hierzu befindet sich in der Anlage.

9. Projekt-Abschluss

9.1 Zeitlicher Soll-/Ist-Vergleich

Abschließend wurde noch ein Vergleich der ursprünglich geplanten Zeiten sowie der tatsächlich aufgewendeten Zeiten durchgeführt. Hierbei stellte sich eine Abweichung bei der Analyse-Phase heraus, der durch den geplanten Puffer in der Implementierungs-Phase kompensiert werden konnte.

Projektphase	Geplante Zeit	Tatsächliche Zeit	Differenz

Analyse	5 Std.	6 Std.	+1 Std.
Entwurf	10 Std.	10 Std.	0 Std.
Implementierung	52 Std.	51 Std.	-1 Std.
Abnahme	3 Std.	3 Std.	0 Std.
Einführung	3 Std.	3 Std.	0 Std.
Dokumentation	7 Std.	7 Std.	0 Std.
Gesamt	**80 Std.**	**80 Std.**	**0 Std.**

TABELLE 5: ZEITLICHER SOLL-/IST-VERGLEICH

Hintergrund der Abweichung in der Analyse-Phase ist, dass die zu Grunde liegenden Textdateien nicht korrekt dokumentiert waren.

9.2 Lessons Learned

Für Vitesco ergibt sich als Ergebnis aus dieser Projekt-Durchführung die Erfahrung, dass individuelle Lösungen zeitnah und effektiv sowie effizient umgesetzt werden können. Auch ist das Feedback hierzu, dass der Einsatz von Praktikanten sich in erheblichen Maß lohnen kann. Insbesondere die sehr schnelle Amortisations-Dauer war ein Signal für Vitesco, zukünftig weitere Projekte durch Praktikanten durchführen zu lassen.

9.3 Persönliches Fazit

Ich selbst konnte bei diesem Projekt feststellen, dass eine solide Projektplanung unerlässlich ist, selbst wenn die eigentliche Programmierung in agiler Art erfolgt. Insbesondere für die lineare Planung und die Erstellung des Pflichtenheftes waren die vielfältigen Erfahrungswerte von Herrn Hagg als Leiter Local IT unerlässlich und ich sehe hier für mich großes Potential, von diesen Erfahrungswerten zu profitieren.

9.4 Ausblick

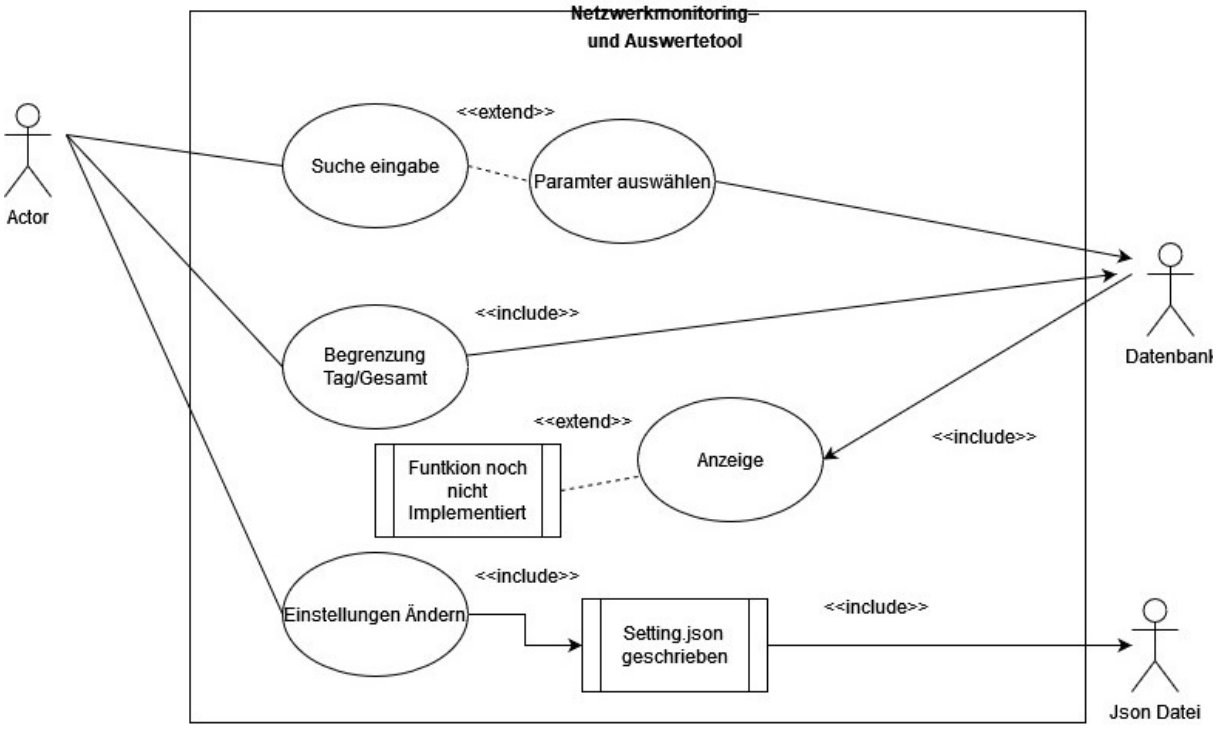
Das Projekt ist ein guter Grundstein für eine Weiterentwicklung. Auch die GUI kann um weitere Diagnose-Funktionen erweitert werden, wie eingangs erwähnt ist z.B. die Implementierung eines selbstlernenden Modells für Predictive Maintanence möglich.

III. Anlagen

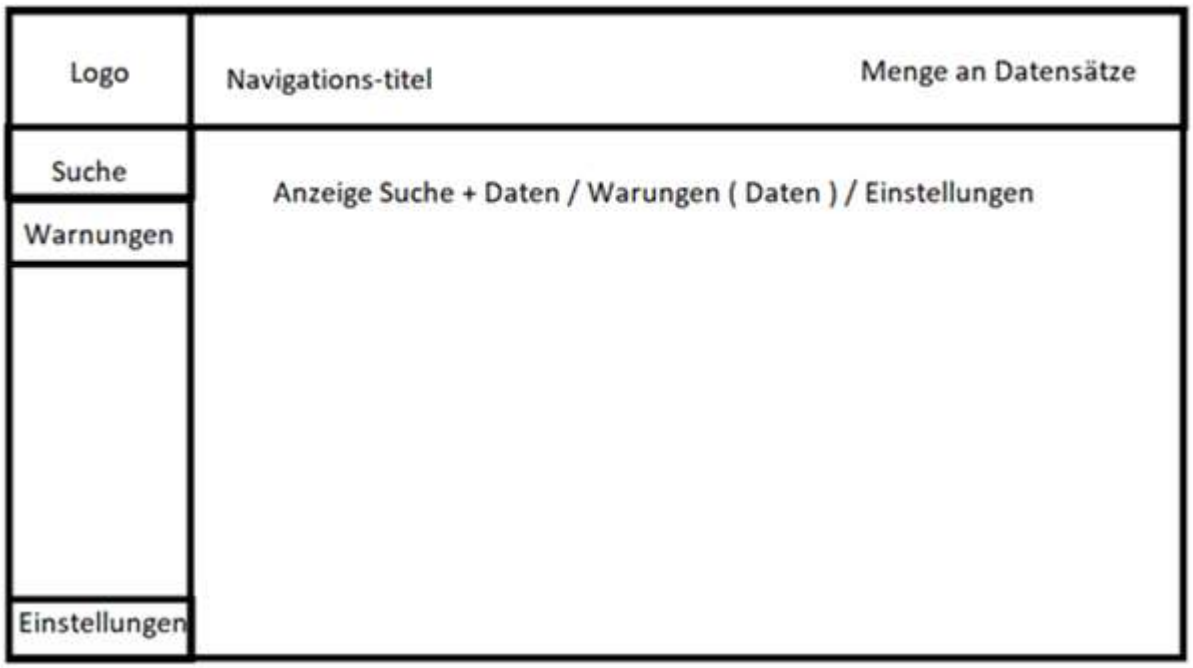
A1 Datenbank-Modell

Clients
MAC_Adresse TEXT
IP_Adresse TEXT
Device_Name TEXT
Device_Loaction TEXT
Nummer TEXT
Switch_Name TEXT
Switch_Port TEXT
Hersteller TEXT
Type TEXT
Geraetetype TEXT
Name TEXT
Datum TEXT

A2 Use-Case-Diagramm



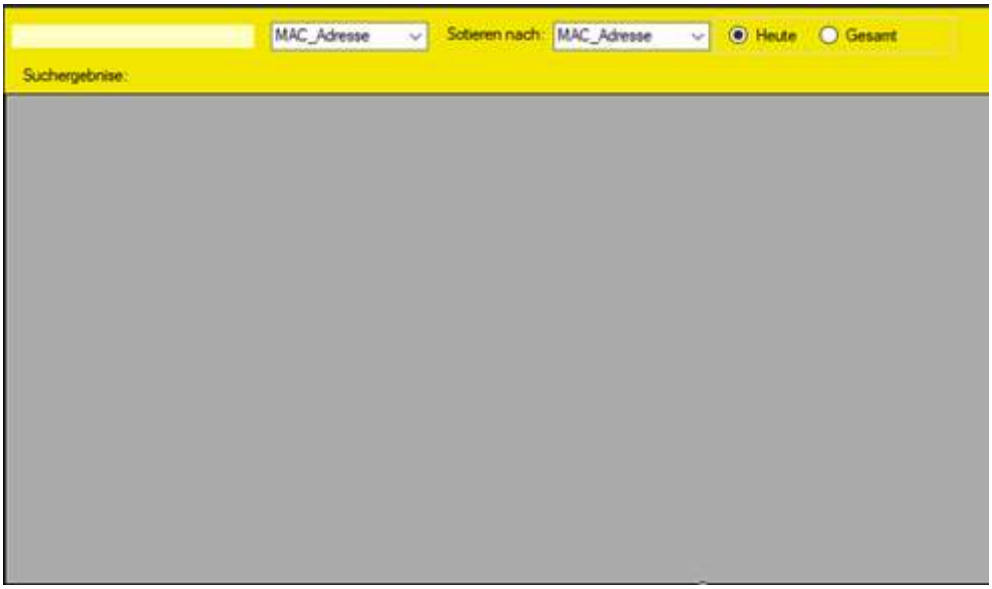
A3 Mockup / Oberflächenentwurf



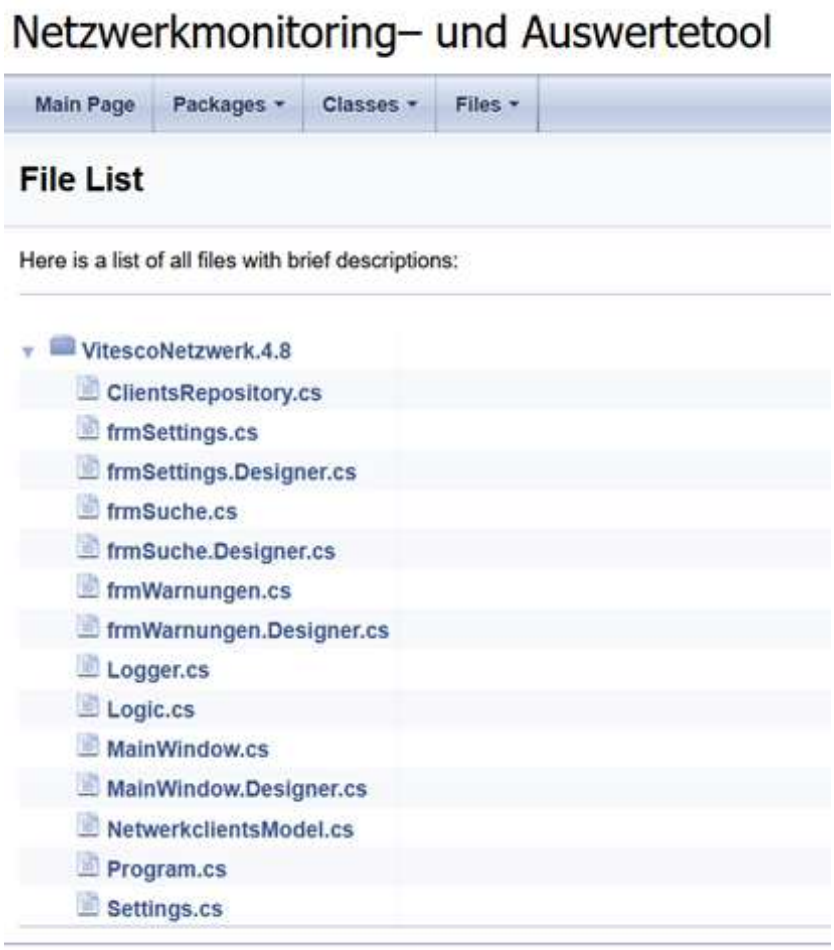
A4 Abbildungen der Benutzeroberfläche

Hinweis: Aus Gründen des Datenschutzes werden in Screenshots keine Daten angezeigt!


vitesco
TECHNOLOGIES
Willkommen


MAC_Adresse
Sotieren nach:
MAC_Adresse
Heute
Gesamt
Suchergebnise:

A5 Entwickler-Dokumentation



A6 Auszug aus dem Lastenheft

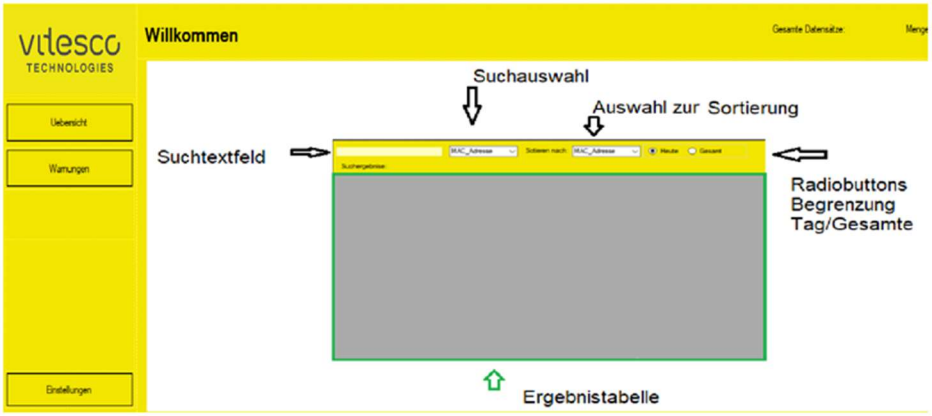
3 Beschreibung der Anforderungen

- Programmierung der Anwendung in C#
- Erstellen einer Datenbank mithilfe von SQLite
- Lauffähigkeit des Programmes unter Windows Server 2019 (.net 4.8)
- Designen einer grafischen Oberfläche zur Anzeige der Datenbank
- Funktion, welche die DB nach entsprechenden Kriterien durchsucht
- Automatische Synchronisation der DB bei Anwendungsstart
- Selbsterklärende Benutzeroberfläche
- Detaillierte Dokumentation der Funktionen

A7 Auszug aus der Benutzerdokumentation

Benutzerhandbuch

Die Übersicht



1 Suchtextfeld

Hier kann ein Attribut gesucht werden z.b.: Macadresse, IP usw.

Dies muss aber Übereinstimmen mit der Suchauswahl!

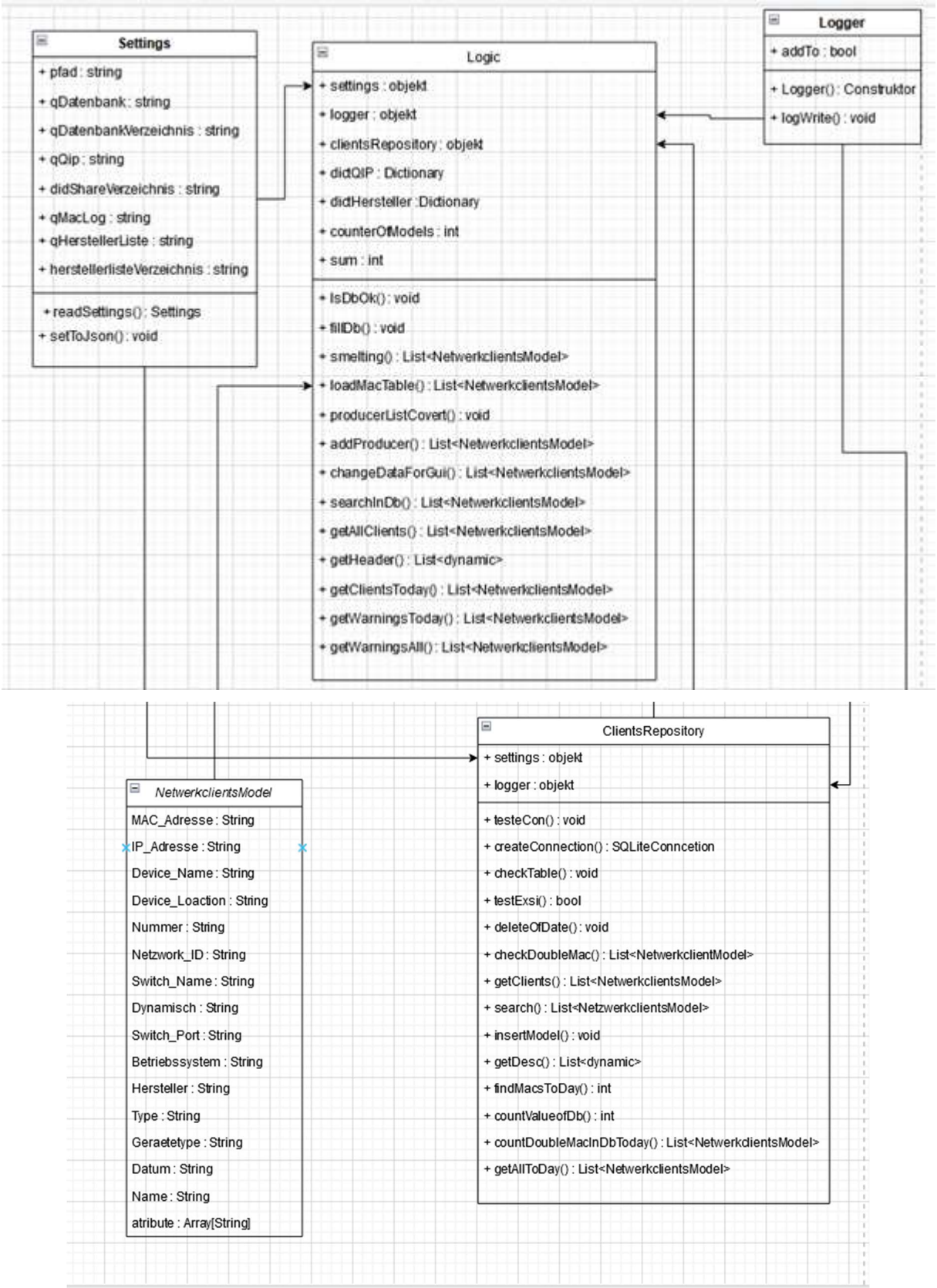
2 Suchauswahl

In diesem Dropdownmenü kann ausgewählt werden nach was gesucht werden soll.

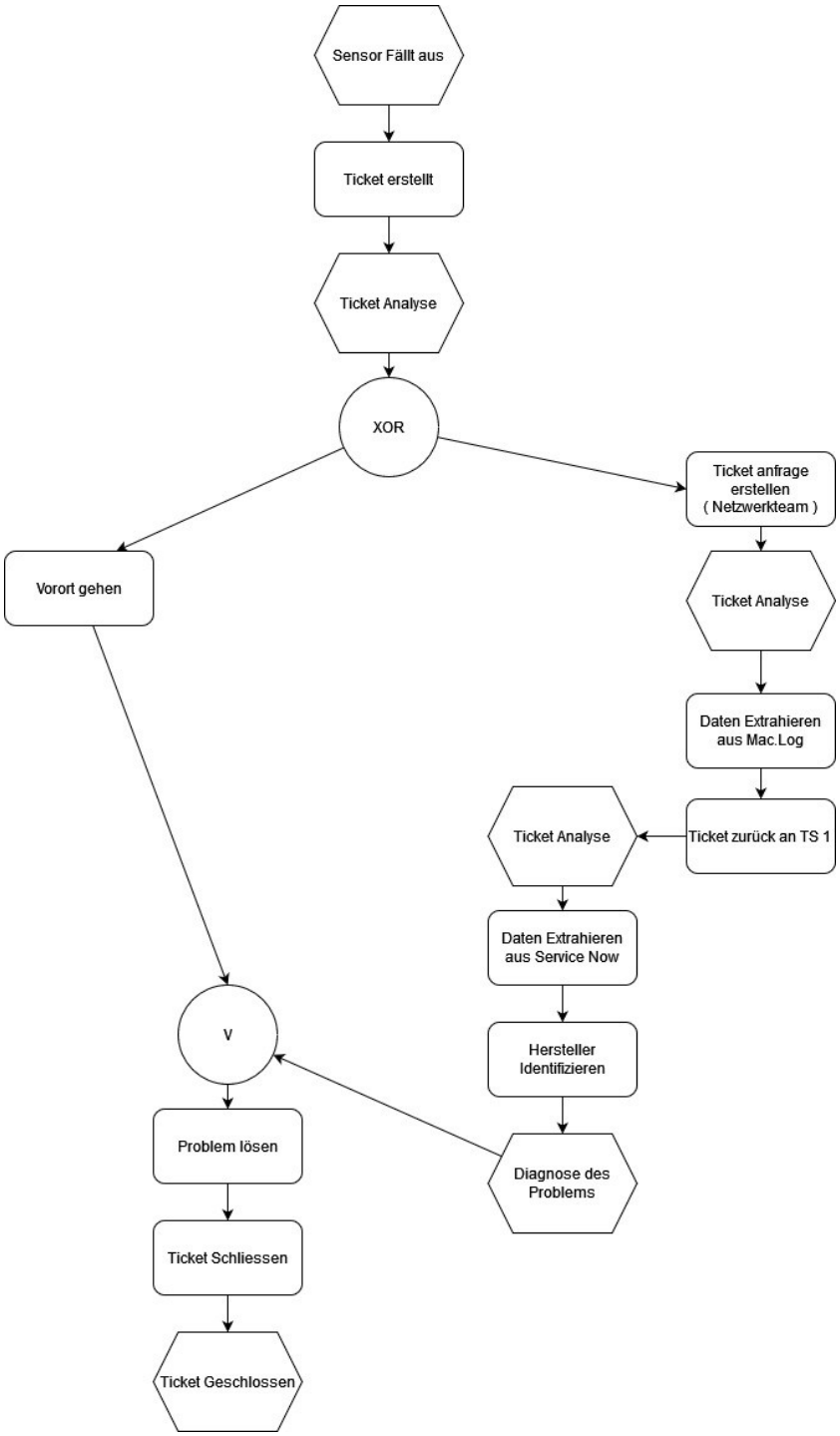
3 Auswahl zur Sortierung

Bei diesem Dropdownmenü handelt es sich um die Sortierung, je nach Auswahl wird die

A8 Klassendiagramm



A9 Ereignisgesteuerte Prozesskette (EPK) VOR der Umsetzung



A10 Auszüge des Quellcodes

```
public List<NetwerkclientsModel> loadMacTable()
{
    // zeitmessung
    Stopwatch stopwatch = Stopwatch.StartNew();
    stopwatch.Start();
    // Lesse die macLogs ein
    string[] zeilen = File.ReadAllLines(settings.qMacLog);


    List<NetwerkclientsModel> list = new List<NetwerkclientsModel>();

    // variablen angelegt
    string NameDesSwitches = "";
    string nummer = "";
    string mac = "";
    string port = "";

    foreach (string zeile in zeilen)
    {
        // zeiteile die Zeile anhand der Kommatars
        string[] zeileA = zeile.Split(',');
        // 1 datum  2 switch ( name ) 3 nummer 4 mac 5. port
        // lösche die sonderzeichen aus der Macadresse
        // verhindert fehler bei der übergabe in die datenbank
        mac = zeileA[3];
        mac = mac.Replace(":", "");
```

```
        mac = zeileA[3];
        mac = mac.Replace(":", "");
        mac = mac.Replace(".", "");
        mac = mac.Replace("-", "");

        port = zeileA[4];
        nummer = zeileA[2];
        NameDesSwitches = zeileA[1];
        string datum = zeileA[0];
        // jede Model  nach der aufbereitung in Liste hinzugefügt
        list.Add(new NetwerkclientsModel
        {
            Nummer = nummer,
            MAC_Adresse = mac,
            Switch_Port = port,
            Datum = datum,
            Switch_Name = NameDesSwitches
        });

    }
    // zeitmessungs ende
    stopwatch.Stop();
    Console.WriteLine("Zeit in der LoadMac-Funktion" + stopwatch.Elapsed);
    return list;
```

```
/// <summary>
/// Diese Funktion bekommt die Liste aus der Maclog und überprüft jedes Model anhand der Macadresse in der QIP.c
/// </summary>
/// <param name="maclog"></param>
/// <returns>Liste NetwerkclientsModel></returns>
public List<NetwerkclientsModel> smelting(List<NetwerkclientsModel> maclog)
{
    // zeitmessung
    Stopwatch stopwatch = Stopwatch.StartNew();
    stopwatch.Start();
    // hier wir die QIP eingelessen
    string[] zeilen = File.ReadAllLines(settings.qQip);
    // für jede mac addresse in meiner Maclogliste
    foreach (NetwerkclientsModel m in maclog)
    {
        foreach (string zeile in zeilen)
        {
            string[] words = zeile.Split(',');
            if (words.Length < 2)
            {
                words = zeile.Split(';');
            }
            if (words.Length < 2)
            {
                Console.WriteLine("Die QIP wurde Verändert ( Name der Datei? )! Trennung nur durch , und ; gestattet.
            }
```

```
            Console.WriteLine("Die QIP wurde Verändert ( Name der Datei? )! Trennung nur durch , und
        }
        //ObjIpAddr,MacAddr,Name,ObjectType,ObjectClass,ObjectDesc,
        //WEn nicht in qip gefunden in warhun
        if (words[0] == "ObjIpAddr")
        {
            continue;
        }
        // wen die mac gefunden würde werden die daten aufgefüllt
        if (words[1] == m.MAC_Adresse)
        {

            m.IP_Adresse = words[0];
            m.Name = words[2];
            m.Type = words[3];
            m.Geraetetype = words[4];
            m.Device_Name = words[5];



        }
    }
}
stopwatch.Stop();
Console.WriteLine("Zeit in der Schmelzen-Funktion " + stopwatch.Elapsed);

return maclog;
```

```
public void producerListCovert()
{
    // zeitmessung
    Stopwatch stopwatch = Stopwatch.StartNew();
    stopwatch.Start();
    // varis
    string HerstellerListe = settings.herstellerlisteVerzeichnis;
    string str;

    // hersteller liste gedownloadet?
    bool herstellerListeDa = File.Exists(HerstellerListe);
    if (herstellerListeDa == false)
    {
        // wen nicht lade wir sie
        WebClient mywebClient = new WebClient();
        mywebClient.DownloadFile(settings.qHerstellerListe, HerstellerListe);

    }
    // lesse die herstelller liste ein
    str = File.ReadAllText(HerstellerListe);

    // ersten 4 zeilen auf den string löschen
    int indexOfFirstMac = str.IndexOf("00");

    // hier schneide ich den string zu um den such bereich zu verkleiner und fehler auszuschliesen
    str = str.Substring(indexOfFirstMac);
```

```
// hier schneide ich den string zu um den such bereich zu verkleiner und fehler auszuschliesen
str = str.Substring(indexOfFirstMac);
str = str.Replace("\n", string.Empty);
str = str.Replace("\r", ",");
str = str.Replace("\t", ",");
str = str.Replace(".", string.Empty);
str = str.Replace("'", string.Empty);

string[] ListeHersteller = str.Split(',');
ListeHersteller = ListeHersteller.Where(x => !string.IsNullOrEmpty(x)).ToArray();

// hier nehme ich die macadresse
for (int a = 0; a < ListeHersteller.Length; a++)
{
    if (ListeHersteller[a].Contains("(base 16)"))
    {
        if (dicHersteller.ContainsKey(ListeHersteller[a].Substring(0, 6)))
        {
            // hier überspringe ich den Header
            dicHersteller[ListeHersteller[a].Substring(0, 6)] = " MEHR EINTRAEGE " + dicHersteller[ListeHersteller[a].Substring(0, 6)];
            continue;

        }
        else
        {
            // und erstelle die disct
```

```
            else
            {
                // und erstelle die disct
                dicHersteller.Add(ListeHersteller[a].Substring(0, 6), ListeHersteller[a + 1]);
            }
        }
    }
    str = null;
    ListeHersteller = null;
    // ende der zeit messung
    stopwatch.Stop();
    Console.WriteLine("Zeit in der herstellerListeCovertieren-Funktion " + stopwatch.Elapsed);


}
```

```
public bool testExsi()
{
    // funktion zum testen ob die datenbank vorhanden ist und automatisches erstellen dieser
    try
    {
        if (File.Exists(settings.qDatenbank) != true)
        {

            Console.WriteLine("Keine Datenbank gefunden! --- Erstelle Datanbank ---");
            SQLiteConnection.CreateFile($"{settings.qDatenbank}");
            Console.WriteLine("Datenbank erstellt!");

            return false;



        }
        //Console.WriteLine("testExsi funktioniert");
        return true;
    }
    catch (Exception e) { logger.logWrite("ClientsRepository - testExist()", e.ToString()); }
    return false;
}
```

```
public void testeCon()
{
    try
    {
        // funktion zum testen der Datenbank verbindung
        SQLiteConnection sqlCon;
        sqlCon = new SQLiteConnection($"Data Source={settings.qDatenbank};Version = 3;New = True;Compress = True;");
        sqlCon.Open();
        sqlCon.Close();
    }
    catch (Exception e) { logger.logWrite("ClientsRepository - testeCon()", e.ToString()); }

}
```

```
public void checkTable()
{
    try
    {
        // funkton zum prüfen und erstellen des Tables in der Datenbank
        NetwerkclientsModel model = new NetwerkclientsModel();

        using (IDbConnection db = createConnection())
        {
            // genau typen definieren
            db.Execute($"CREATE TABLE IF NOT EXISTS 'Clients' ( {model.atribute[0]} TEXT, " +
            $"{model.atribute[1]} TEXT, {model.atribute[2]} TEXT, {model.atribute[3]} TEXT, {model.atribute[4]} TEXT," +
            $"{model.atribute[5]} TEXT, {model.atribute[6]} TEXT , {model.atribute[7]} TEXT, {model.atribute[8]} TEXT," +
            $"{model.atribute[9]} TEXT, {model.atribute[10]} TEXT , {model.atribute[11]} TEXT );");

        }
        //Console.WriteLine("checkTable funktioniert");
    }
    catch (Exception e) { logger.logWrite("ClientsRepository - checkTable()", e.ToString()); }

}
```

```
/// <summary>
/// Diese Funktion fürt die anderen funktionen zum zusammen führen der Daten aus und erstellt eine Liste aus NetzwerkclientsModel
/// </summary>
public void fillDb()
{

    // hier wird eine zeitmessung gestartet
    Stopwatch stopwatch = Stopwatch.StartNew();
    stopwatch.Start();

    Console.WriteLine("Fülle Datenbank");
    // erstelle die macliste
    List<NetwerkclientsModel> Mac = loadMacTable();
    // füge beide listen zusammen ( qip und mac )
    List<NetwerkclientsModel> ErsteListe = smelting(Mac);

    // jetzt werden die hersteller raus gesucht und in die liste übertragen
    List<NetwerkclientsModel> ListeNachAbgleich = addProducer(ErsteListe);

    this.sum = ListeNachAbgleich.Count;

    // da die liste nur zur Db übergeben werden kann
    // für jedes model in meiner liste
    foreach (NetwerkclientsModel model in ListeNachAbgleich)
    {
```

```
    foreach (NetwerkclientsModel model in ListeNachAbgleich)
    {
        try
        {
            // verusche ich einen Inster in die datenbank
            // frage an mich selbst.. nicht gleich die ganze liste?
            clientsRepository.insert(model);
            // kontrollzähler um zu sehen das das programm nicht hängt
            this.counterOfModels = this.counterOfModels + 1;
            Console.WriteLine(counterOfModels);
        }
        // das hier ist klar fals fehler kommt
        catch (Exception e)
        {
            Console.WriteLine(e.Message);

        }
    }
    Console.WriteLine("Füllung beendet..");
    // ende der zeitmessung und ausgabe
    stopwatch.Stop();
    Console.WriteLine("Zeit in der fillDB-Funktion " + stopwatch.Elapsed);
    Console.WriteLine("Prüfe Ob Datenbank vollständig");
}
```

```
/// <summary>
/// Diese Funktion Prüft auf vollständige übertragen in die Datenbank anhand der Maclog und den heutigen einträgen in der Datenbank
///
/// </summary>
public void isDbOk()
{

    // zeitmessung
    Stopwatch stopwatch = Stopwatch.StartNew();
    stopwatch.Start();

    List<NetwerkclientsModel> maclogs = loadMacTable();

    string datum = maclogs[0].Datum;

    // eine prüfung auf die länge und vollständigkeit der übertragung in die datenbank


    // so ist der zeitpunkt in der db 31.01.2023_14:15:03
    Console.WriteLine("Beginne Prüfung");
    int anzahlInMaclog = 0;
```

```
int anzahlInMaclog = 0;


// zähle die maclogs ab nach datum
for (int i = 0; i < maclogs.Count; i++)
{
    if (maclogs[i].Datum == datum)
    {
        anzahlInMaclog = anzahlInMaclog + 1;
    }

}


Console.WriteLine("Anzahl der macs im maclog");
Console.WriteLine(anzahlInMaclog);
Console.WriteLine(datum);
// anzahl der neuen Macadressen in der datenbank heute
int anzahlInDB = clientsRepository.findMacsToDay(datum);


Console.WriteLine("Anzahle der macs in der Datenbank vom datum der Maclog");
Console.WriteLine(anzahlInDB);
// hier vergleiche ich die anzahl der macadressen in der Maclog mit der aus der Datenbank
while (anzahlInMaclog > anzahlInDB)
{
```

```
while (anzahlInMaclog > anzahlInDB)
{
    // wen die maclog anzahl gröser ist fehlt was in der Datenbank
    Console.WriteLine("Unterschied festgestellt!");
    Console.WriteLine("Räume Datenbank einträge auf");
    // lösche die vorhandenen einträge und
    clientsRepository.deleteOfDate(datum);
    // fülle die datenbank neu
    fillDb();
    anzahlInDB = clientsRepository.findMacsToDay(datum);
}
    // hier sind alle macaddressen in der datenbank
    Console.WriteLine("Keine Differenz in der Datenbank festgestellt!");
// durch den selbst aufruf übergebe ich 2 uhrzeiten, die maclogs besteht aus
// 2 identischen einträgen mit einer zeit differenz von 1 sekunde
// natürlich prüfe ich beide auf vollständigkeit
stopwatch.Stop();
Console.WriteLine("Zeit in der IsDbOk()-Funktion " + stopwatch.Elapsed);


#endregion
```

IV Abkürzungs-Verzeichnis

AG	Aktiengesellschaft
CSV	Comma Separated Value (Siehe Glossar)
IEEE	Institute of Electrical and Electronics Engineers (Siehe Glossar)
DIE	Integrated Development Enivironment (Siehe Glossar)
GUI	Graphical User Interface (Siehe Glossar)
Json	JavaScript Object Notation
ORM	Objektrelationale Abbildung (Siehe Glossar)
MAC	Media-Access-Control-Address
IP	Internet Protocol

V Glossar

Hinweis: Sortiert nach dem jeweils ersten Auftreten in der Dokumentation.

Vitesco	Der Auftraggeber
Predictive Maintenance	Eine auf Basis von selbstlernenden Modellen „vorausschauende Wartung“
Local IT	Die Fachabteilung für die direkte IT-Betreuung von Vitesco
CSV- Datei	Eine kommagetrennte Datei zur Beschreibung von Tabellen-Daten
IEEE	Eine Organisation, die sich mit der Standardisierung und Organisation elektrische und elektronischer Voraussetzungen beschäftigt
SQLite	Ein Datenbank-System, das direkt in ein Programm compiliert werden kann, so dass das Pogramm selbst zum Datenbank-Server wird
IDE	Eine Entwicklungsumgebung, die grafische und textuelle Bestandteile einer Software berücksichtigt
GUI	Die grafische Oberfläche einer Anwendung
Git	Ein System zur Versionierung von Quellcodes, das gleichzeitig den Review von Code-Bestandteilen ermöglicht
Json	Eine Notations-Form, in der sich serialisierte Objekte als Text-Datei darstellen lassen
ORM	Objektrelationale Abbildung (englisch object-relational mapping, ORM).Eine Technik der Softwareentwicklung, mit der ein in einer objektorientierten Programmiersprache geschriebenes Anwendungsprogramm seine Objekte in einer relationalen Datenbank ablegen kann.
SCRUM	Ein Entwicklungs-Modell auf agiler Basis, abweichend zum linearen Phasen-Modell.
Review	Die retrospektive Betrachtung eines Quellcodes zur Qualitätssicherung
Log-Datei	Eine Datei, die spezielle Informationen zur Protokollierung enthält
Ressourcen-Kosten	Begleitende Kosten einer Entwicklung, zum Beispiel Miete, Strom, Lizenzen,

Amortisation	Gemeint ist der Zeitpunkt, ab dem sich Aufwand und Nutzen eines Projektes überschneiden und das Projekt „Gewinn“ erwirtschaftet.
Usability	Die Benutzerbarkeit einer Software, also z.B. die sinnvolle grafische Gestaltung der Oberfläche
Access-Point	Jeder beliebige Punkt, über den auf ein Netzwerk zugegriffen werden kann
Funktion	Jeder Programmcode, der ein Ergebnis liefert
Klasse	Der Bauplan für ein Objekt
Objekt	Die Instanz einer Klasse