Der TitelPflichtenheft und technische Spezifikation im   
Programmierprojekt

**Datenanalyse mit neuronalen Netzwerken**

Mitarbeiter:

Lukas Evers

Anderson Lewis Orock Soh Talla

Paul Schult

Versionshistorie

|  |  |  |
| --- | --- | --- |
| Datum | Bearbeiter | Durchgeführte Änderung |
| 20.10.20 | PS | Erstellung |
|  |  |  |

Inhaltsverzeichnis

[1 Visionen und Ziele 1](#_Toc5699678)

[2 Anforderungen an Ihr System 1](#_Toc5699679)

[2.1 Use-Cases 1](#_Toc5699680)

[2.2 Aktivitäten-Diagramme 1](#_Toc5699681)

[2.3 GUI 1](#_Toc5699682)

[3 Realisierung 2](#_Toc5699683)

[3.1 Allgemeines 2](#_Toc5699684)

[3.2 Interne Schnittstellen 2](#_Toc5699685)

[3.3 Visual-Studio-Projektsetup 2](#_Toc5699686)

[3.4 Externe Schnittstellen 3](#_Toc5699687)

[4 Test und Implementierungsphase 3](#_Toc5699688)

[5 Planung 3](#_Toc5699689)

[6 Lizenz 4](#_Toc5699690)

# Visionen und Ziele

Ziel dieses Projekts ist die Erstellung einer Desktop-Anwendung, die mit skalaren Trainingsdaten ein Machine Learning Modell trainiert und testet. Anschließend kann das damit erstellte Modell verwendet werden, um anhand neuer Daten das zu erwarteten Ergebnis zu schätzen.

*Desktop-Anwendung, mit der beliebige csv-Dateien eingelesen werden können.*

*Beschreiben Sie die Visionen und Ziele für das Programmierprojekt. Stellen Sie dabei die wesentlichen Dinge des Lastenheftes dar und beschreiben Sie aus Anbietersicht den Nutzen, der mit dem zu realisierenden System erzielt werden soll.*

*Hier zeigt der Anbieter, dass er das Problem des Auftraggebers verstanden hat und die einzelnen Punkte auch in der richtigen Priorisierung erfasst hat.*

*Ziel ist es eine Anwendung zu schreiben, welche eine beliebige Exceltabelle (mit Überschriften) einlesen kann. Das Programm soll die Möglichkeit bieten auf Basis der Tabelle ein Machine Learning Modell (z.B. Neuronales Netzwerk) zu trainieren und zu testen. Dabei muss in der Anwendung die Zielspalte ausgewählt werden (welche Variable soll das ML-Modell überhaupt schätzen). Weiterhin muss eine Auswahl von Eingabedaten (Spalten in der Tabelle) durch den Benutzer erfolgen.*

*Die Anwendung soll das Modell mittels der zusätzlichen Nutzereingaben automatisch lernen und testen. Zusammen mit dem Kunden wählen Sie entsprechende Testdaten. Im Gegensatz zum verwandten Thema in diesem Semester liegt hier der Fokus klar auf der Anwendung und nicht auf der Analyse unterschiedlicher Varianten von ML-Modellen.*

|  |  |  |  |
| --- | --- | --- | --- |
| Feature-ID | Priorität | Beschreibung | Aufwand |
| F.100 |  | Desktop-Anwendung |  |
| F.200 |  | Daten vorbereiten |  |
| F.201 |  | CSV Dateien einlesen |  |
| F.202 |  | Datenbereich und Zielspalte bestimmen |  |
| F.300 |  | Neuronales Netzwerk trainieren |  |
| F.400 |  | Neuronales Netzwerk testen |  |
| F.401 |  | Ergebnis-Sicherheit bestimmen |  |
| F.402 |  | Neuronales Netzwerk speichern |  |
| F.500 |  | Neuronales Netzwerk einsetzen |  |
| F.501 |  | Ergebnis in CSV Datei ausgeben |  |
| F.5z1 |  | Ergebnis auf dem Bildschirm ausgeben |  |
|  |  |  |  |

# Anforderungen an Ihr System

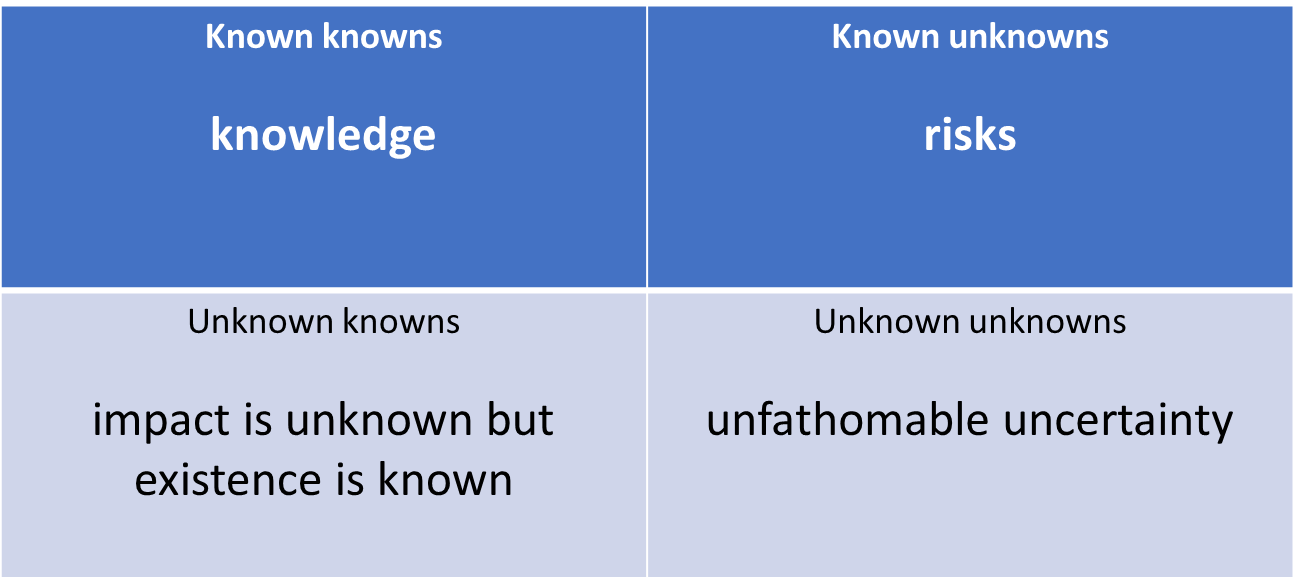
## Use-Cases

Beschreiben Sie die wesentlichen Use-Cases Ihres Systems mit einem (oder mehreren) Use-Case-Diagrammen. Zeigen Sie im Diagramm evtl. vorhandene Subsysteme und ordnen Sie die Use-Cases den betreffenden Subsystemen zu.

Legen Sie fest, welches Teammitglied für welches Use-Case verantwortlich ist.

## Risiken

Ordnen Sie Ihre Use-Cases und Technologien in folgendes Schema ein:



|  |  |
| --- | --- |
| Known knowns   * Termin Pflichtenheft * Termin funktionaler Prototyp * Termin Version 1.0 * GUI Programmierung * Projektmanagement | Known unknowns   * Fehlende Erfahrung mit ML.NET |
| Unknown knowns   * Anwendbares Wissen anderer bekannter Programmiersprachen (C++, Python etc.) | Unkown unknowns   * Während Phase 2 und 3 Komponentenübergreifende Änderungen am Projekt (CommonInterface) |

## GUI

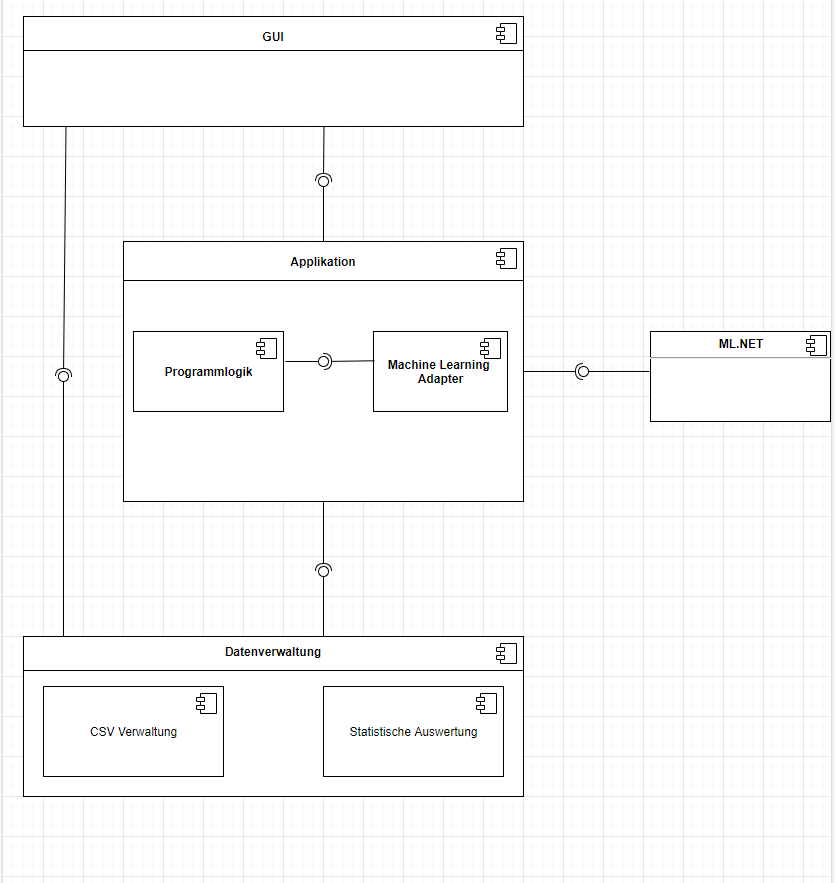
Erstellen Sie einen Mockup Ihrer GUI. Dazu sollen für die wichtigen Anwendungsfälle die Oberflächen entworfen und ihre Funktion beschrieben werden.

# Realisierung

## Allgemeines

Legen Sie die für Ihr System zu erstellenden Komponenten fest. Ordnen Sie die Komponente dem verantwortlichen Teammitglied zu. Legen Sie fest, welche Use-Cases in welcher Komponente oder welchen Komponenten aus heutiger Sicht zu realisieren sind.

Erstellen Sie ein Komponenten-Diagramm des zu realisierenden Programms mit den dazu erforderlichen Komponenten und zeigen Sie die Abhängigkeiten zwischen den Komponenten auf.



## Interne Schnittstellen

Erstellen Sie eine Liste mit allen zu realisierenden Schnittstellen gemäß des Komponentendiagramms. Überlegen Sie, welche Funktionen von welcher Komponente bereitgestellt werden müssen und definieren Sie damit alle Schnittstellen in einem Klassendiagramm.

Erstellen Sie danach eine Visual-Studio-Projektmappe und fügen Sie ein neues Klassenbibliotheks-Projekt hinzu, das Sie CommonInterfaces nennen. Erstellen Sie darin nun alle gelisteten Interfaces, dem Sie einen mit „I“ beginnenden und darauf folgend den Namen der bereitstellenden Komponente geben.

Gäbe es bspw. eine Komponente UserManagement im Komponentendiagramm, die von anderen Komponenten benutzt würde, so hieße das entsprechende Interface IUserManagement. In der später anzulegenden Komponente (Projektmappe) UserManagement muss dann eine Klasse dieses Interface implementieren.

In den Interfaces müssen bereits die in der Liste genannten Methoden und Eigenschaften enthalten sein, die den anderen Komponenten bereitgestellt werden sollen. Die Funktionsparameter und Rückgabewerte sollen schon die inhaltlich wichtigen Dinge abbilden.

Denken Sie daran, dass Interfaces während des Projektes weitgehend stabil bleiben sollten. Stecken Sie also vorab genügend Überlegungen in diesen Punkt, um ihre Teamkollegen nicht ständig mit Änderungen zu nerven!

Alternativ zu Interfaces können Sie in den CommonInterfaces auch abstrakte Klassen verwenden, wenn dies funktional geboten ist.

|  |  |  |
| --- | --- | --- |
| Schnittstelle | Methoden | Eigenschaften |
| ICSVHandling | load(); safe(); |  |
| IStatistics |  |  |
| IMLAdapter |  |  |
| ILogic | loadcsv(Dateipfad);savecsv();splitcsv();trainModel();testModel();  predictandReturnResult();loadmodel(); savemodel();amountofcolumns();setColumnsTyp(); gettenRows(); evalutateModel(); |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Visual-Studio-Projektsetup

Ergänzen Sie nun Ihre Visual-Studio-Projektmappe um C#-Projekte für alle definierten Komponenten, legen Sie dabei den Typ des Projektes korrekt fest (entweder Windows-Anwendung oder Klassenbibliothek).

Für eine normale Desktop-Anwendung gibt es nur eine Applikation (Exe bzw. Windows Anwendung), alle anderen Komponenten sind Dlls (Klassenbibliotheken), nur die später noch anzulegenden Testprojekte sind ebenfalls Executables.

Sofern Sie UI-Elemente in Ihrem Prototyp haben, erstellen Sie das in 2.3 vorgeschlagene Mockup (noch ohne Funktion) für Ihren Prototyp.

Ergänzen Sie Ihre Komponenten/Visual-Studio-Projekte um Klassen, die die in 3.2 definierte Schnittstelle implementieren bzw. von der abstrakten Klasse erben und geben Sie der Klasse einen zweckmäßigen Namen. Hinweis: alle Klassenbibliotheken (bis auf die CommonInterfaces) müssen eine Klasse beinhalten, die ein Interface implementiert oder ein Interface einer anderen Komponente benutzen.

Checken Sie die Projektmappe mit allen Projekten und Test- bzw. Dummy-Projekten in git ein.

## Externe Schnittstellen

Beschreiben Sie die externen Schnittstellen Ihres Systems. Erstellen Sie dazu eine Tabelle, in der Sie den Namen der Schnittstelle, die Art der Schnittstelle (das könnte eine Datei, ein anderes System sein), den Typ der Schnittstellenimplementierung (Dateizugriff, http-Zugriff, COM-Zugriff o.ä.) und die Komponente, die für die Realisierung der externen Anbindung zuständig ist, auflisten.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Art | Typ | Realisiert in Adapter | Herausgeber |
| Microsoft.ML/ML.NET | Bibliothek | Dateizugriff? | MLAdapter | Microsoft |
|  |  |  |  |  |
|  |  |  |  |  |

# Test und Implementierungsphase

Erstellen Sie in Ihrer Projektmappe Test-Projekte, um die Komponenten (Dlls) individuell entwickeln und testen zu können. Jedes Teammitglied soll exakt 1 eigenes Executable-Projekt haben, mit dem die eigenen Komponenten getestet werden können. Für die Entwicklung der GUI stellt die GUI selber das Executable dar, dann allerdings mit Dummy-Komponenten die die später einzubindenden Komponenten der Teammitglieder imitieren und exakt die gleichen Schnittstellen haben, aber nur Testdaten liefern. Denken Sie daran, die Interfaces auch in den Dummy-Klassen zu verwenden, sofern Ihre Komponente derartige Dummies benötigt.

Bestehen also Abhängigkeiten zu Komponenten anderer Teammitglieder, sind dafür eigene Dummykomponenten zu entwickeln, die die definierte Schnittstelle implementieren und Dummy-Daten liefern bzw. nichts tun, wenn z.B. eine Komponente für das Speichern zuständig ist. In der letzten Projektphase werden diese Dummy-Dlls dann durch die fertigen Dlls der Teammitglieder ersetzt.

Wenn die eigene Komponente eine Klassenbibliothek ist, muss ein Testprojekt als Windows-Anwendung erstellt werden, das die eigene Komponente einbindet und dann testen kann. Dies kann mit einer GUI oder auch als Konsolenanwendung realisiert werden.

Der Projektsetup hat also so zu erfolgen, dass jedes Teammitglied eine eigene Testumgebung hat, ohne Verweise auf Komponenten anderer Mitglieder zu haben. Diese Verweise sind durch die beschriebenen Dummy-Projekte zu ersetzen. Dazu können in svn für jedes Teammitglied zusätzlich zur gesamten Projektmappe eigene Projektmappen im git angelegt werden.

Jedes Teammitglied erstellt zusätzlich ein Komponentendiagramm des eigenen Projektsetups. Somit sind insg. N+1 Komponentendiagramme abgegeben, eines für jedes Mitglied und eines für das fertige Programm.

# Lizenz

Legen Sie im Team fest, ob ihre erstellte Software anschließend open source ist bzw. von wem sie nachgenutzt werden kann. Eine übliche Lizenz für Hochschulprojekte ist die [MIT-Lizenz](https://de.wikipedia.org/wiki/MIT-Lizenz).