**[*Intelligente Parkplatzerkennung mit künstlichen neuronalen Netzwerken*]**

**Anforderungsdokument**

[1. Ziel und Zweck des Dokumentes 3](#_Toc3914326)

[1.1.1. Projektbeschreibung 3](#_Toc3914327)

[1.1.2. Kurzbeschreibung des Projekts 3](#_Toc3914328)

[1.1.3. Zweck des Projekts 3](#_Toc3914329)

[1.1.4. Hintergrund, Problemstellung, Motivation für das Projekt 4](#_Toc3914331)

[1.1.5. Ziele des Projekts 4](#_Toc3914340)

[1.1.6. Erfolgskriterien 4](#_Toc3914341)

[2. Anforderungen an die Dokumentation der Anforderungen, Use Cases etc. 5](#_Toc3914342)

[2.1. Definitionen 5](#_Toc3914343)

[2.2. Anforderungen an die Dokumente der Anforderungsdokumentation selbst 5](#_Toc3914344)

[2.3. Use Cases 6](#_Toc3914345)

[2.4. Szenarien 8](#_Toc3914346)

[2.5. User Stories 8](#_Toc3914347)

[2.6. Software Stories 9](#_Toc3914348)

[2.7. Standardformat der Anforderungsdokumentation 9](#_Toc3914349)

[2.7.1. Beispiel: 13](#_Toc3914350)

[2.7.2. Weiteres Beispiel 13](#_Toc3914351)

[3. User Stories / Use Cases und Szenarien /Software Stories der … 15](#_Toc3914352)

[4. Anforderungen an … 16](#_Toc3914353)

[5. Anhang 17](#_Toc3914354)

[5.1. Abkürzungen 17](#_Toc3914355)

[5.2. Glossar 17](#_Toc3914356)

[5.2.1. Definition der Objekte 17](#_Toc3914357)

[5.2.2. Definition der Verben 17](#_Toc3914358)

[5.3. Literatur 17](#_Toc3914359)

[5.4. Bildverzeichnis 18](#_Toc3914360)

[5.5. Tabellenverzeichnis 18](#_Toc3914361)

Versionen:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rev. | Datum | Autor | Bemerkungen | Status |
| 0.1 | 14.03.2019 | Felix Willrich | 1. Entwurf + Eintragen aller Informationen | Abgeschlossen |
| 0.2 | 19.03.2019 | Felix Willrich | Beschreibung des Projekts eingetragen | In Bearbeitung |

1. Ziel und Zweck des Dokumentes

Dieses Dokument beschreibt die Anforderungen der T-Systems on site services GmbH. Es handelt sich hierbei um die Systemdefinition, die der Auftragnehmer für den Auftraggeber (Kunde) erstellt, sodass der Kunde versteht und validieren kann, was das System leisten wird.

Projektbeschreibung

Das Projekt umfasst eine Erkennung von freien Parkplätzen mithilfe von maschinellem Lernen. Dazu sollen verschiedene Techniken und Inputdaten genutzt werden, um eine möglichst hohe Genauigkeit zu erzielen.

Kurzbeschreibung des Projekts

Das Projekt folgt einem gewissen Ablauf. Die Daten werden eingelesen, verarbeitet und ausgegeben. Diese drei Schritte werden anhand von folgendem Bild verständlich.

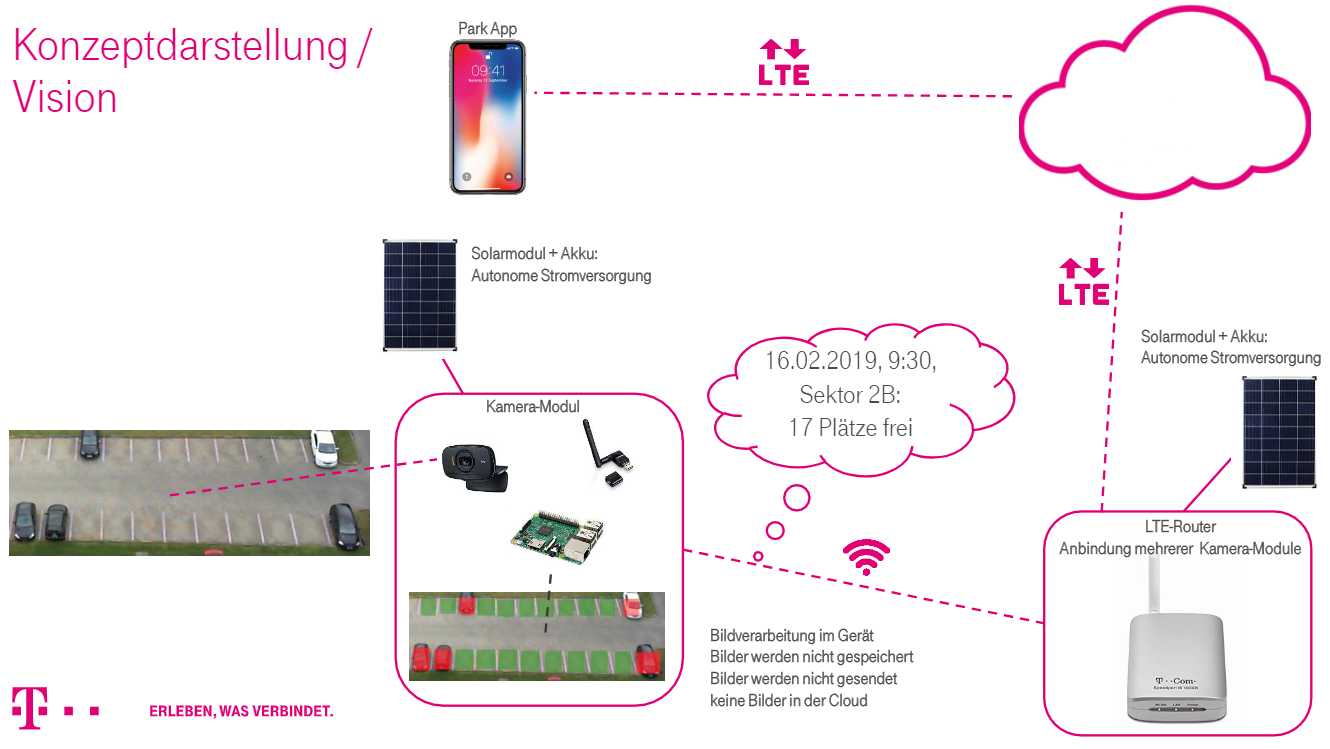


Abbildung : Produkt-Zyklus

Eine Kamera überträgt ein regelmäßig aufgezeichnetes Bild von einem Parkplatz an einen Mikrocontroller. Dieser verarbeitet die Daten, indem er das Bild in das konzipierte neuronale Netzwerk schickt. Anhand der Aufnahme soll bestimmt werden, wie viele Parkplätze frei sind. Diese Information wird an eine zentrale Stelle geschickt und weiterverteilt an eine Smartphone-App, damit die Nutzer zu jeder Zeit abrufen können, wohin sie fahren sollten. Die Bilder werden nicht vom Mikrocontroller weitergeschickt.

Zweck des Projekts

Das Projekt soll in erster Instanz zur Erkennung von freien Flächen eingesetzt werden und im Rahmen von Parkplätzen und Parkhäusern genutzt werden. Die Technik kann mit verschiedenen Inputdaten auf verschiedenste Felder ausgeweitet werden. Beispiele wären, Lagerbestände oder den Füllstand von verschiedenen Containern erkennen.

Hintergrund, Problemstellung, Motivation für das Projekt

Die T-Systems on site services GmbH in Person von Philip May benutzt im produktiven Sektor verschiedene Machine Learning/Deep Learning Applikationen und möchten durch dieses Produkt in weitere Felder stoßen bzw. weitere Erkenntnisse darüber gewinnen.

Für die Gruppe ergibt sich aufgrund von wenig Vorkenntnissen folgende Probleme:

* Neues Umfeld kennen lernen
* Geeignete Tools und Umgebung finden
* Datenbeschaffung zum Anlernen
* Prototypen erschaffen
* Genaue Erkennung implementieren
* Testumgebung

Die meisten Probleme werden oder wurden mit unserem Ansprechpartner besprochen und

teilweise aufgearbeitet.

Die Motivation zu diesem Projekt ergibt sich aus dem ersten Stichpunkt der Probleme. Die Gruppe möchte in ein neues, aufstrebendes und sehr interessantes Thema einsteigen und dabei gleichzeitig Praxiserfahrung sammeln.

Ziele des Projekts

Das Projekt wird zuerst bis zu dem Schritt entwickelt bis das Netz angelernt ist

und verschiedene Parkplatzsituationen erkannt werden. Eine Genauigkeit von 99% wird

angestrebt.

Erfolgskriterien

Sollte die gewünschte Genauigkeit erlangt worden sein, wird das Projekt als Erfolg bezeichnet.

Weiterhin hinzukommen würden verschiedene Umgebungen, wie Schnee, Regen und andere Hindernisse wie Baulöcher oder Belegung von zwei Parkplätzen gleichzeitig. Sollten diese zusätzlichen Kriterien erfüllt werden, wird das Projekt in vollem Umfang als Erfolg gewertet. Es wird eine möglichst genaue Erkennung mit allen unterschiedlichen Faktoren angestrebt.

1. Anforderungen an die Dokumentation der Anforderungen, Use Cases etc.

In diesem Kapitel finden Sie nochmals Wiederholungen einiger Aspekte, die der Erstellung von Anforderungen hilfreich sein können, siehe hierzu auch Software-Technik von Prof. Müller, z.B. [3]

## Definitionen

Unter *funktionalen* bzw. *nicht-funktionalen Anforderungen* wollen wir gemäß [5] (Abschn. 4.1) Folgendes verstehen:

Bei den **funktionalen Anforderungen** handelt sich hier um Dienstleistungen (Services), die das zu entwickelnde System bereitstellt, d.h. wie das System sich bei bestimmten Eingaben und in bestimmten Situationen verhält. Funktionale Anforderungen können genauso gut beschreiben, was das System **nicht** machen soll.

Bei **nicht-funktionalen** Anforderungen handelt es sich um Randbedingungen, die für die Funktionen (Services) des zu entwickelnden Systems bestehen. Dieses sind hier z.B.

* Randbedingungen für die Laufzeit und den Speicherbedarf,
* Randbedingungen für die Zuverlässigkeit,
* Randbedingungen an die Benutzerfreundlichkeit des Systems,
* Randbedingungen für den Entwicklungsprozess,
* Randbedingungen, die aus Standards oder Gesetzen resultieren.

**Extra-funktionale Anforderungen** beziehen sich häufig auf das System als Ganzes und nicht so sehr auf einzelne Funktionalitäten. Sie haben damit meist großen Einfluss auf die Gesamtarchitektur des Systems und nicht nur auf einzelne Komponenten.

Extra-funktionale Anforderungen sind für den Erfolg des Systems aus Kunden- oder Benutzersicht meist viel entscheidender. Die Benutzer finden häufig Wege, um fehlerhafte oder fehlende Funktionen irgendwie zu umschiffen. Eine nicht umgesetzte extra-funktionale Anforderung kann dazu führen, dass das gesamte System unbrauchbar wird.

In der Realität ist die Unterscheidung zwischen diesen beiden Arten nicht immer so eindeutig möglich, wie es hier die Definitionen suggerieren. Nichtsdestotrotz werden wir es hier versuchen. Wir werden daher funktionalen und extra-funktionalen Anforderungen kennzeichnen.

## Anforderungen an die Dokumente der Anforderungsdokumentation selbst

Die Anforderungen müssen **vollständig** sein, d.h. alle Dienstleistungen, die vom Auftraggeber gewünscht werden, müssen sich auch in den dokumentierten Anforderungen widerspiegeln.

Die Anforderungen müssen **konsistent** sein, d.h. sie dürfen sich untereinander nicht widersprechen.[[1]](#footnote-1)

Nicht-funktionale Anforderungen sollten – wann immer möglich – quantitativ formuliert sein. Beispiel:

Die Anforderung „Die SBU muss einen Plan in akzeptabler Zeit anzeigen“ ist zu schwammig. Besser wäre: „Die SBU muss einen Plan für 40 Gäste nach höchstens 5 Sekunden Rechenzeit anzeigen.

Im Idealfall beschreiben die Anforderungen „nur“ das externe Verhalten und die operationellen Randbedingungen des Systems. Sie beschreiben nicht, wie das System entworfen oder implementiert wird. Praktisch ist das aber bei komplexen Systemen aus den folgenden Gründen so gut wie unmöglich, weil:

* Eine erste Architekturvorstellung die Strukturierung der Anforderungen erleichtert.
* Das zu entwerfende System muss mit anderen Systemen zusammenarbeiten, sodass hierdurch das Design eingeschränkt wird, d.h. hieraus Anforderungen resultieren.
* Die Verwendung einer bereits erprobten Architektur kann vorgeschrieben sein.

Die Anforderungen werden hier in natürlicher Sprache beschrieben. Allerdings sollen alle Anforderungen einem Standardformat genügen.

Es ist zu unterscheiden zwischen zwingenden Anforderungen (mandatory requirements) und Anforderungen (desirable requirements), die möglichst oder wenn möglich umgesetzt werden sollen.

Die folgenden Folienkopien wurden aus dem Skript von Prof. Müller [3] übernommen und finden sich so auch in dem Lehrbuch von Rupp et al. [4].

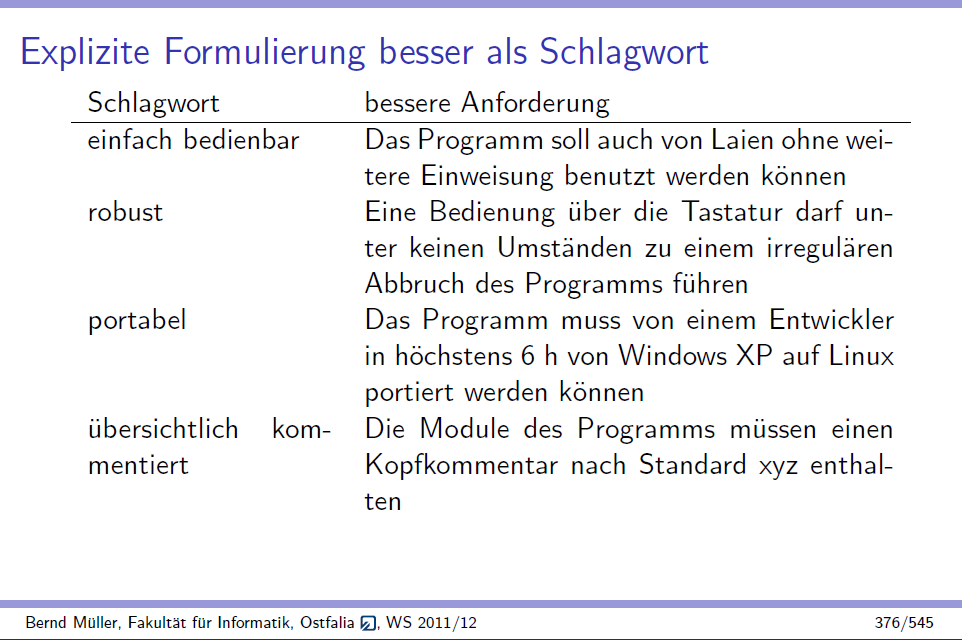


Abbildung 2 Vermeidung von Schlagworten

## Use Cases

Use Cases sind Bestandteil von UM zur Beschreibung der Interaktion von Akteuren mit dem System. Es gibt keine strenge Unterscheidung zwischen Use Cases und Szenarien. Mehrere Use Cases können durch ein Szenario beschrieben werden und umgekehrt. Man kann vielleicht sagen, dass Use Cases eine andere Möglichkeit sind, das Systemverhalten durch Beispiele zu beschreiben.

Ein Use Case beinhaltet immer den Aktor, der in die Interaktion eingebunden ist und einen kurzen Namen für die Interaktion. Der Use Case wird dann noch durch zusätzliche Informationen ergänzt. Dieses kann formal und informell erfolgen.

Aktor und Interaktion werden durch (gerade) Linien miteinander verbunden. Es können auch Pfeile verwendet werden, wenn ausgedrückt werden soll, wo die Interaktion beginnt/initiiert wird. Sie beginnt am Pfeilanfang.

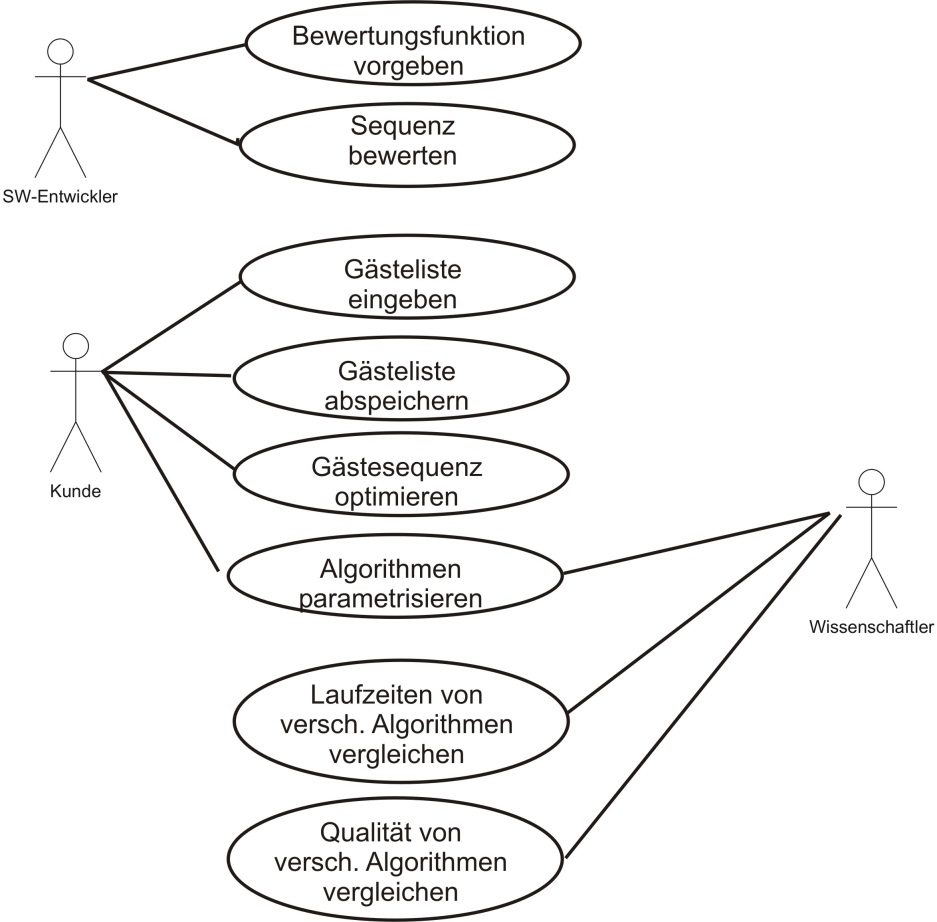


Abbildung 3 Einige Use Case der Scheduling Bewertungsumgebung

Für die Erstellung von Use Cases können Sie auch eine Schablone verwenden, z,B.

|  |  |
| --- | --- |
| Identifier | SBU-U-03 |
| Name | Anwendungsfälle in Datenbank verwalten |
| Beschrei  bung | Das System verwaltet Anwendungsfälle in einer Datenbank |
| Beteiligte Akteure | System |
| Status | bereit zum Review |
| Verwendete Use Cases | Keine |
| Auslöser | Ein Anwendungsfall soll von dem System verarbeitet werden, ODER ein Anwendungsfall wurde von dem System verarbeitet. |
| Vorbedingungen | Das Programm ist gestartet.  Die Art des Anwendungsfalls wurde in das System gelesen oder ausgewählt. |
| Invarianten | Das Programm läuft weiter. |
| Nachbedingung / Ergebnis | Das Programm arbeitet weiter. |
| Standardablauf | Das System lädt und speichert Anwendungsfälle in einer Datenbank. |
| Hinweise |  |
| Historie: | |  |  |  | | --- | --- | --- | | 22.02.15 | HHe | Initiale Formulierung des Use Cases | |  |  |  | |  |  |  | |  |  |  | |

## Szenarien

Formale und abstrakte Anforderungen sind insbesondere für die Endanwender schwer zu verstehen und damit für ihn auch schwer zu beschreiben oder zu kritisieren. Anwender kommen besser mit Beispielen aus ihrem Alltag zurecht, d.h. diese sind einfacher zu verstehen und zu kritisieren.

Jedes Szenario beschreibt normalerweise eine oder wenige mögliche Interaktionen des Benutzers mit dem System, die er im Rahmen eines Use Cases durchführt. Requirements Engineers benutzen die Szenarien meist, um daraus die Anforderungen abzuleiten.

Nach [5] (Abschn. 4.5.3) beinhaltet ein Szenario:

1. Eine Beschreibung, was das System bzw. der Benutzer zu Anfang erwarten (die Ausgangssituation),
2. Eine Beschreibung des normalen Ablaufs des Szenarios,
3. Eine Beschreibung, was schiefgehen kann und wie das System damit umgeht,
4. Zusatzinformation zu Aktivitäten, die während des Szenarios ebenfalls stattfinden können.
5. Eine Beschreibung des Systemzustands, wenn das Szenario beendet ist (Endzustand).

## User Stories

**User Stories** (Anwendererzählung) wie sie im Rahmen von SCRUM oder eXtreme Programming verwendet werden, sind in der Alltagssprache formulierte Anforderungen zur Planung von Sprints. Meist sind es nicht mehr als ein bis zwei Sätze. User Stories. Eine gute Einführung in diese Thematik findet sich z.B. im Internet bei Ambler [1] .

Es gibt Parallelen zu den Use Cases (Anwendungsfällen) der UML. Anwendungsfälle werden in der Regel detaillierter und mit mehr Kontextinformation beschrieben.

* Beispiel: Wenn der Anwender die Scheduling-Bewertungsumgebung (SBU) startet, kann er zunächst zwischen drei Optionen auswählen: 1. Alle Tests durchführen, 2. Ein Scheduling-Problem lösen, 3. Zufällige Erzeugung von Scheduling-Problemen.
* Beispiel: Als Kunde des Online-Buchhandels möchte ich Bewertungen anderer Käufer lesen, um mich für oder gegen den Kauf eines Buches entscheiden zu können.

User Stories können entweder formlos oder etwas formaler erstellt werden. Es bietet sich insbesondere für den Anfänger an, die folgende Schablone zu verwenden:

Als <Rolle>, möchte ich <<Ziel/Wunsch><, um <<Nutzen>.

User Stories sind ein Planungsinstrument, um in einem agilen Entwicklungsprozess zu entscheiden, was im nächsten Zeitraum umgesetzt werden soll. Die Angabe des Nutzens hilft dabei, die Priorität einer User Story festzulegen. User Stories müssen im Rahmen der Planung so weit zerlegt werden, bis der Aufwand zur Umsetzung geschätzt werden kann und sie in einem Implementierungszeitraum umgesetzt werden können.

User Stories werden meist handschriftlich auf Karteikarten notiert. Man versucht gleich eine Priorität und eine Bearbeitungsdauer zu schätzen. User Stories geringer Priorität werden in der Regel nicht so detailliert wie solche mit hoher Priorität beschrieben. Erst wenn ihre Priorität ausreichend groß wurde, sodass sie für die Bearbeitung ausgewählt wurden, werden sie weiter detailliert. Dafür bieten sich die Rückseiten der Karteikarten an.

## Software Stories

Software Stories [4] sind konkrete, beispielhafte Szenarien, die den Ablauf einer User Story oder eines Use Cases veranschaulichen. In einer Software Story werden – wie in einem Storyboard die einzelnen Aktivitäten, Teilfunktionen oder Prozessschritte in Form eines konkreten Beispielablaufs mit beispielhaften Daten skizziert. In [4] wird eine graphische Notation ergänzt um textuelle Erklärungen vorgeschlagen. Software Stories sind eine einfache Möglichkeit, Szenarien zu modellieren, die einzelnen Aktivitäten und ihre Reihenfolge zu spezifizieren, ihre Eingaben und Ausgabe und die beteiligten Stakeholder zu identifizieren. Auch Bedingungen und Sonderfälle können behandelt werden. Aus den beispielhaften Daten wird das Datenmodell einer Anwendung hergeleitet. Ein ausführliches Beispiel findet sich in [4].

## Standardformat der Anforderungsdokumentation

Verwenden Sie zur Formulierung von Anforderungen die Schablone von Rupp et al. [4].

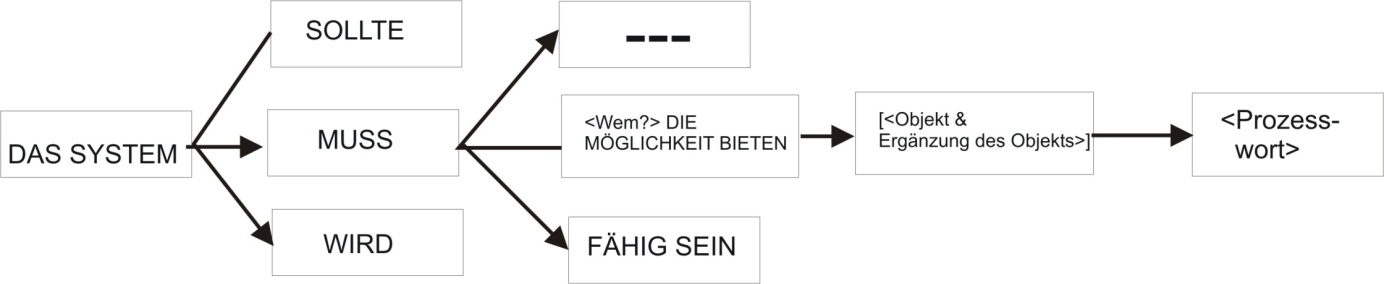


Abbildung 4 Schablone für Anforderungen [4]

Beispiel 1:

Die SBU sollte dem Anwender die Möglichkeit bieten, die Badbelegung mit dem Greedy-Algorithmus zu optimieren.

Für SYSTEM wird jeweils der Name des zu entwickelnden Systems eingesetzt, in unserem Fall somit „Die SBU“. Die Wichtigkeit der Anforderung wird durch MUSS, SOLLTE oder WIRD ausgedrückt. MUSS legt eine rechtliche Verbindlichkeit fest. Die Erfüllung der Anforderung ist verpflichtend und somit kann die Abnahme des Produktes durch den Kunden verweigert werden. SOLLTE spezifiziert wünschenswerte Eigenschaften. Sie sind nicht verpflichtend. Ihre Erfüllung erhöht aber signifikant die Zufriedenheit des Kunden. WIRD weist auf Anforderungen hin, die in der Zukunft integriert werden. Dann sind sie verpflichtend. Diese Anforderungen helfen, in der aktuellen Lösung Vorbereitungen zu treffen, um Zukünftiges später leichter integrieren zu können.

Der dritte Teil in Abbildung 3 legt die Art der Systemaktivität fest. Bei einer selbstständigen Aktivität des Systems bleibt der Teil leer. Tritt der Benutzer mit dem System in Interaktion, um eine Funktionalität zu starten, verwenden wir die Formulierung DIE MÖGLICHKEIT BIETEN. Liegt dagegen eine Schnittstellenanforderung vor, wird FÄHIG SEIN verwendet. Im Unterschied zur Benutzeraktion führt unser System hier nur in Abhängigkeit von einen Dritten (System) eine Aktion aus.

Das Prozesswort legt die geforderte Funktionalität fest, d.h. was das System durchführen soll, z.B. drucken, speichern etc. Verwenden Sie hier auf jeden Fall Vollwerben und keine Hilfsverben. Verwenden Sie so wenig verschiedene Verben wie möglich. Legen Sie zuvor im Glossar die Bedeutung dieser Verben fest. Beschreiben Sie auch alle Synonyme dieses Verbs. Verwenden Sie diese aber nicht, sondern möglichst nur das eine Vollverb.

Objekt & Ergänzung des Objekts spezifiziert das Prozesswort näher. Es wird das WAS und WO des Prozesswortes näher spezifiziert.

Beispiel 2:

Die SBU muss dem Anwender die Möglichkeit bieten, einen Anwendungsfall zur Badbelegungsplanung in einer ASCII-Datei in einem vordefinierten Format einzufügen.

* MUSS: Es handelt sich um eine verbindliche Anforderung
* dem Anwender die Möglichkeit bieten: Benutzerinteraktion
* Anwendungsfall zur Badbelegung ist noch näher zu spezifizieren. Hier ist z.B. das Glossar ein geeigneter Ort. Dieser Begriff wird bestimmt auch noch an anderer Stelle benötigt. Dabei kann auch gleich das vordefinierte ASCII-Dateiformat zur Badbelegungsplanung beschrieben werden.
* einfügen ist das Prozesswort: Dieses solle im Glossar auch geeignet definiert werden, z.B.  
  In der SBU muss „*einfügen*“ den Prozess bezeichnen, neue Daten einzugeben oder vorhandene Daten zu überschreiben.

„vordefinierte ASCII-Dateiformat zur Badbelegungsplanung:“

Die ASCII-Datei ist wie folgt aufgebaut:

* <beschreibende Textzeile>
* <Gast> <Gast> ... <Gast>

<Gast> besteht aus <Name> <Aufstehzeit> <Badzeit> <Planzeit>.

<Aufstehzeit> ist eine <Uhrzeit>.

<Uhrzeit> ist <Ziffer2>:<Ziffer2>.

< Ziffer2> besteht aus einer oder zwei Ziffern.

usw.

An irgendeiner Stelle wird man aufhören, zumindest mit dem Detailgrad, den man noch mit dem Kunden diskutieren will, denn oft schließen sich – salopp gesagt –noch mit dem Kunden diskutieren will, denn oft schließen sich – salopp gesagt – Verständlichkeit und Korrektheit aus.

Etwas formaler – zumindest für den eigenen Gebrauch – könnten Sie im Glossar auch durch Verwendung der Backus-Naur-Form das Format festlegen:

<vordefinierte ASCII-Dateiformat zur Badbelegungsplanung> ::=  
 <beschreibende Textzeile> <Gäste>

<Gäste> ::= <Gast> [<Gast>]<<

<Gast> ::= <Name> <Aufstehzeit> <Badzeit> <Planzeit><

<Aufstehzeit> ::= <Uhrzeit><

<Badzeit> ::= <Uhrzeit><

<Uhrzeit> ::= <Ziffer2> ":" <Ziffer2><

<Ziffer2> :: = <Ziffer> [<Ziffer>]<

<Ziffer> ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"

usw.

Beispiel 3:

Die SBU sollte die Badbelegung mit dem Greedy-Algorithmus optimieren.

oder

Die SBU sollte jede (in ihr modellierte) Anwendungsdomäne mit dem Greedy-Algorithmus optimieren.

* *SOLLTE* legt fest, dass die Anforderung nicht verpflichtend ist (die Optimierung kann auch mit einem anderen Algorithmus) durchgeführt werden.
* *optimieren* ist hier das Prozesswort, das im Glossar näher definiert werden sollte.  
  In der SBU muss optimieren den Prozess bezeichnen, systematisch durch möglichst wenig Schritte eine Reihenfolge der Elemente zu finden, sodass eine vorgegebene Bewertungsfunktion ihren minimalen Wert erreicht, sofern genügend Rechenzeit und Speicherplatz zur Verfügung steht. Sind Rechenzeit und/oder Speicherplatz begrenzt kann der Prozess auch vorher (in einem lokalen Minimum) terminieren.[[2]](#footnote-2)

Diese Anforderung könnte aber auch als Benutzerinteraktion formuliert werden:

Die SBU sollte dem Anwender die Möglichkeit bieten, die Badbelegung mit dem Greedy-Algorithmus zu optimieren.

Nicht jede Anforderung muss immer erfüllt sein, sondern sie gilt nur unter gewissen Bedingungen. Bedingungen sind normalerweise zeitlich oder logische. Zeitliche Bedingungen werden z.B. mit den Schlüsselworten wenn, solange oder nachdem eingeleitet. Logische Bedingungen beginnen mit „falls“.

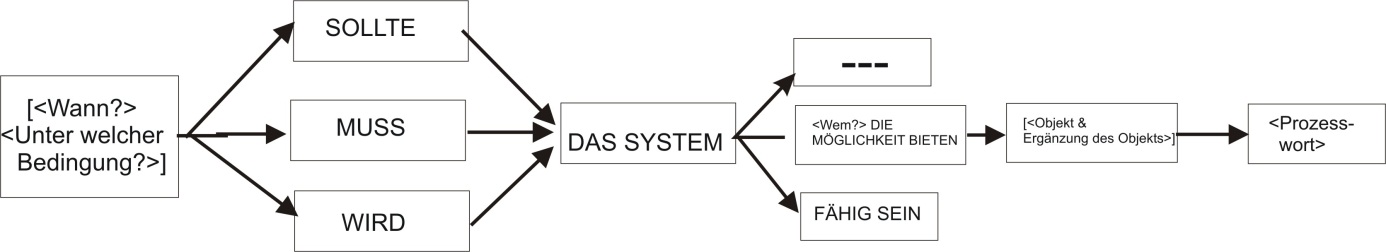
Wie Abbildung 4 zeigt, wird die Schablone aus Abbildung 3 lediglich am Anfang um die Bedingung erweitert und im Deutschen wird das Hilfsverb dem SYSTEM vorangestellt. Bei komplexeren Bedingungen bietet es sich an, die Anforderung inklusive der Bedingungen in mehrere Sätze zu zerlegen.

Abbildung 5 Schablone für bedingte Anforderungen [4]

Beispiele:

Solange die vom Anwender vorgegebene maximale Laufzeit noch nicht überschritten ist, sollte die SBU die bis zu diesem Zeitpunkt gefundene beste Lösung weiter optimieren.

Falls eine Methode mehr als 10 Zeilen Code umfasst, muss die Scheduling-Bewertungsumgebung für diese Methode in mindestens einen JUnit-Test getestet werden.

Falls bei der Badbelegungsplanung des ersten Gastes bekannt ist, dass sich bereits ein Gast im Bad befindet und er noch weitere X > 0 Minuten im Bad benötigt, so muss der folgende Gast mindestens noch X Minuten warten, bis er das Bad betreten darf.

Sie hätten z.B. für das zweite Beispiel auch schreiben können:

Für eine Methode, die mehr als 10 Zeilen Code umfasst, muss die Scheduling-Bewertungsumgebung für diese Methode in mindestens einen JUnit-Test getestet werden.

Hier wurde auf das Bedingungsschlüsselwort „Falls“ verzichtet. Hier vergibt man sich allerdings die Chance, explizit auf das Vorhandensein einer Bedingung hinzuweisen.

Und noch einige weitere Tipps

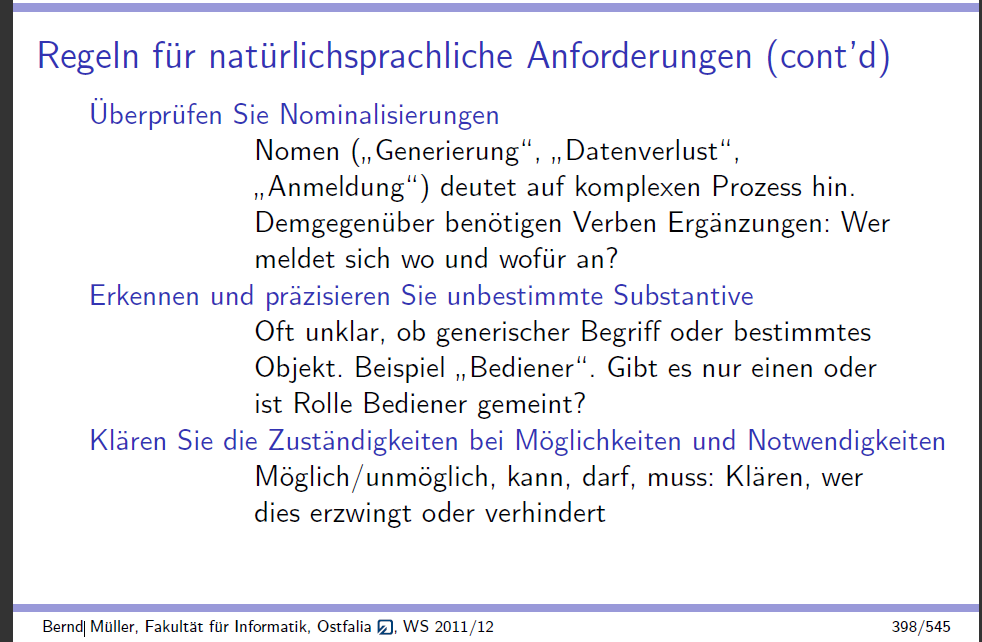


Abbildung 6 Tipps für natürlich sprachliche Anforderungen

Eine Anforderung besteht aber nicht nur aus diesem einen Satz, sondern zu einer Anforderung, wie wir sie hier verwenden wollen, enthält zusätzlich noch:

* eindeutige ID
* Anforderungstyp, d.h. funktionale oder nicht funktionale Anforderung bzw. irgendwie beides
* Nummer der Use Cases (Szenarien), aus der diese Anforderung abgeleitet wurde
* die (bedingte) Anforderung selbst, entsprechend der obigen Schablone
* eine Begründung, warum diese Anforderung erforderlich ist, und was eine Implementierung für positive Effekte hat und/oder welche negativen Effekte eine Nicht-Realisierung dieser Anforderung hätte.
* Abnahmekriterium: Hiermit muss es möglich sein zu testen/zu beweisen, dass eine Implementierung diese Anforderung exakt umgesetzt hat
* Anforderer: Von wem kommt die Anforderung
* Kundenzufriedenheit, wenn diese Anforderung umsetzt ist (groß, normal, klein)
* Priorität der Anforderung (aus Kundensicht), Zahlen von eins bis zehn, oder hoch, normal, niedrig
* Konflikte, wenn man schon weiß, welche Anforderung hiermit im Konflikt steht
* Weitere Erklärungen
* Historie, wann hat wer was an dieser Anforderung geändert

Beispiel:

|  |  |
| --- | --- |
| ID | SBU-Bad-01-04 |
| Anforderungstyp | Funktionale Anforderung |
| Use Case: | Vorgänger beachten |
| Anforderung: | Falls bei der Badbelegungsplanung des ersten Gastes bekannt ist, dass sich bereits ein Gast im Bad befindet und er noch weitere X > 0 Minuten im Bad benötigt, so muss der folgende Gast mindestens noch X Minuten warten, bis er das Bad betreten darf. |
| Begründung: | Die SBU soll nicht nur einmalig vor dem Aufstehen des ersten Gastes eine optimale Lösung liefern können, sondern nachdem neue Eingangsdaten vorliegen, soll abermals optimiert werden können. Die neuen Daten könnten z.B. um 8:20 Uhr sein, dass Rüdiger schon im Bad ist und noch 6 weitere Minuten braucht. Man könnte nun zwar bei jedem Gast die früheste mögliche Zeit auf 08:26 Uhr setzen, aber dieses könnte zu anderen Bewertungen und anderen optimalen Sequenzen führen, zumindest dann wenn keine lineare Aufsummierung der Wartezeiten der Gästen erfolgt. |
| Abnahmekriterium: | Es wird eine Eingabedatei mit den Gästen Rüdiger und Hans und einer frühesten Badeintrittzeit von 08:18 (Rüdiger) und 08:28 (Hans) und Badbelegungszeiten von 8 bzw. 14 Minuten vorgegeben. Außerdem sei Hannes schon seit 08:20 im Bad und brauche noch 6 Minuten. Die SBU muss dann als beste Reihenfolge liefern.   * Rüdiger Eintritt 08:26 (Wartezeit von 8 Minuten) * Hans Eintritt 08:34 (Wartezeit von 6 Minuten)   Für die optimale Sequenz ergibt sich somit eine minimale Wartezeit von 14 Minuten. |
| Anforderer: | Hartmut Helmke |
| Kundenzufriedenheit: | Normal (wird einfach vom Kunden erwartet) |
| Priorität: | Sehr hoch |
| Konflikte: | Derzeit nicht bekannt |
| Weiteres: |  |
| Historie: | |  |  |  | | --- | --- | --- | | 01-01-12 | HHe | Erste Version | | 21-07-12 | HHe | Abnahmekriterium detailliert | | 22-09-12 | SU | Kunde ändert Priorität von normal auf sehr hoch | | 25-02-13 | HHe | 4 durch 6 Minuten im Abnahmekriterium ersetzt (Flüchtigkeitsfehler) | |

Weiteres Beispiel

|  |  |
| --- | --- |
| Anforderung: | Es sollen für jede Verspätung oder negative Abweichung vom geplanten Landezeitpunkt Strafzeiten auf Basis des Quadrats der Zeitdifferenz in Sekunden errechnet werden. |
| Begründung: | Um die Optimierung bewerten zu können, muss eine numerische Bewertungsgrundlage geschaffen werden. |
| Abnahmekriterium: | Strafzeiten werden nach folgender Formel[[3]](#footnote-3) errechnet: |

Ab hier sind nun Sie dran!

1. User Stories / Use Cases und Szenarien /Software Stories der …

*Klatschen Sie bitte nicht User Stories/ Use Cases einfach untereinander, sondern ordnen Sie nach sinnvollen Kriterien (alles zu Optimierungsalgorithmen, alles, an dem Akteur 1beteiligt wird,…)*

*<Falls Sie planorientiert entwickeln, folgen hier ausführliche Use Cases und Szenarien, wobei mindestens die Use Cases für den ersten Meilenstein hinreichend detailliert beschrieben werden müssen.*

*Falls Sie nach Scrum entwickeln, werden hier User Stories erwartet, die den initialen Anforderungsstack ergeben. Wählen Sie für den ersten Release einige zentrale User Stories aus. Diese höchstprioren User Stories müssen über 2-3 Ebenen verfeinert werden, bis Sie zu einer Funktionalität gelangen, die Sie überblicken können.*

*Weniger wichtige User Stories dürfen auf der „High-Level-Ebene“ belassen werden, bis sie zur Realisierung anstehen. Szenarien können ein sinnvolles Hilfsmittel sein bei der Zerlegung der User Stories.>*

*Und jetzt geht es los…*

*Sie können hier frei oder auch durch Verwendung einer Schablone formulieren.*

|  |  |
| --- | --- |
| Identifier | xxx-U-00 |
| Name | Xx |
| Beschreibung | Der Benutzer xxx |
| Beteiligte Akteure |  |
| Status | X |
| Verwendete Use Cases |  |
| Auslöser |  |
| Vorbedingungen | X |
| Invarianten | Das System läuft weiter. |
| Nachbedingung / Ergebnis |  |
| Standardablauf |  |
| Hinweise | Hauptfunktion von xxx |
| Historie: | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |

1. Anforderungen an …

*<Falls Sie nach Scrum vorgehen, erstellen Sie bitte mindestens aus den höchtprioren User Stories, die Sie hinreichend verfeinert haben, detaillierte Anforderunge. Ziel ist, dass Sie daraus Testfälle ableiten können.*

*Falls Sie planorientiert arbeiten, sind mindestens die funktionalen Anforderungen für den ersten Meilenstein detailliert zu beschreiben.*

*Bei allen Vorgehensmodellen sind Datenanforderungen zu beschreiben und die für das ganze Projekt durchgehend wichtigen extrafunktionalen Anforderungen zu dokumentieren.*

*Ordnen Sie Ihre Anforderungen inhaltlich und klassifizieren Sie sie gemäß der folgenden Typen>*

Funktionale Anforderungen

Datenanforderungen

Extrafunktionale Anforderungen

Technische Randbedingungen

Performanz- und Lastanforderungen

Wartungs und Support Anforderungen

Sicherheitsanforderungen (Dependability und IT Security)

Usability Anforderungen

Look and Feel Anforderungen (Style-Vorgaben)

Betriebs und Umgebungsanforderungen

Rechtliche Anforderungen

Vorschlag Schablone

|  |  |
| --- | --- |
| ID | YYY-xxx |
| Anforderungstyp |  |
| User Story/Use Case: | Den hier stehenden Use Case/ User Story muss es dann auch im vorherigen Kapitel geben. |
| Anforderung: |  |
| Begründung: |  |
| Abnahmekriterium: |  |
| Anforderer: |  |
| Kundenzufriedenheit: |  |
| Priorität: |  |
| Konflikte: |  |
| Weiteres: |  |
| Historie: | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |

1. Anhang

## Abkürzungen

Abk. Abkürzung

FH Fachhochschule

## Glossar

Definition der Objekte

Ffff Hier sollten Sie auch etwas reinschreiben

Definition der Verben

Ffff Hier sollten Sie auch etwas reinschreiben

## Literatur

1. Scott Ambler: Introduction to User Stories. Initial User Stories. Artikel in Englisch, http://www.agilemodeling.com/artifacts/userStory.htm; abgerufen am 20. Januar 2012
2. Jochen Ludewig, Horst Lichter:   
   *Software Engineering*,   
   dpunkt Verlag, 2010
3. Bernd Müller: *Software-Technik, Vorlesungsskript*, Fakultät für Informatik, Ostfalia, Hochschule Braunschweig/Wolfenbüttel, Wintersemester 2011/12
4. Ulrich Norbisrath, Albert Zündorf, Tobias George, Ruben Jubeh, Bodo Kraft und Sabine Sachweh: *Software Stories Guide*, Technical Report, Universität Kassel, 2017, https://www.researchgate.net/publication/318826633\_Software\_Stories\_Guide
5. Chris Rupp & die SOPHISTen:   
   *Requirements-Engineering und –Management — Professionelle, iterative, Anforderungsanalyse für die Praxis*,   
   Hanser Verlag, 2009
6. Ian Sommerville:   
   *Software Engineering*,   
   9. Auflage in Englisch. Addison-Wesley, 2010
7. James & Suzanne Robertson:   
   Volere Requirements Specification  
   *Template* Edition 13, August 2007; http://www11.informatik.unierlangen.de/Lehre/WS0809/PR-SWE/Material/volere-template.pdf; abgerufen am 02. Februar 2012

## Bildverzeichnis

[Abbildung 1: Produkt-Zyklus 2](#_Toc3914165)

[Abbildung 2 Vermeidung von Schlagworten 5](#_Toc3914166)

[Abbildung 3 Einige Use Case der Scheduling Bewertungsumgebung 6](#_Toc3914167)

[Abbildung 4 Schablone für Anforderungen [4] 8](#_Toc3914168)

[Abbildung 5 Schablone für bedingte Anforderungen [4] 10](#_Toc3914169)

[Abbildung 6 Tipps für natürlich sprachliche Anforderungen 11](#_Toc3914170)

## Tabellenverzeichnis

1. In der Praxis ist bei größeren und/oder komplexen Systemen quasi unmöglich, die Anforderungen vollständig und widerspruchsfrei niederzuschreiben. [↑](#footnote-ref-1)
2. Durch diese Definition wird quasi festgelegt, dass Optimieren gar nicht das Finden des (globalen) Optimums verlangt, sondern ein ganz gutes Ergebnis reicht auch schon aus. [↑](#footnote-ref-2)
3. Das ist eigentlich die Anforderung. Das Abnahmekriterium beschreibt, wie ein Test zu erstellen ist. Es hilft hier zwar für den Test, aber der wird dann garantiert scheitern, weil die Formel gar nicht in der Anforderung steht. Das Abnahmekriterium sollte vielmehr eine ganz spezifische Instanz der Anforderung enthalten. [↑](#footnote-ref-3)