- Systemanalyse -

Lastenheft für „Lernsoftware zum Verstehen und Programmieren von Turing-Maschinen“

Version: 1.0

|  |  |  |
| --- | --- | --- |
| Projektbezeichnung | tmSim | |
| Projektleiter | Tobias Lettner | |
| Verantwortlich | Team A | |
| Erstellt am | 24.03.2022 | |
| Zuletzt geändert | **26.04.2022 19:35** | |
| Bearbeitungszustand |  | in Bearbeitung |
| **X** | vorgelegt |
|  | fertig gestellt |
| Dokumentablage | **In Git-Branch** | |

Änderungsverzeichnis

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Änderung | | | Geänderte Kapitel | Beschreibung der Änderung | Autor | Zustand |
| Nr. | Datum | Version |
| 1 |  | 1.0 | Alle | Initiale Produkterstellung |  |  |

Prüfverzeichnis

Die folgende Tabelle zeigt einen Überblick über alle Prüfungen – sowohl Eigenprüfungen wie auch Prüfungen durch eigenständige Qualitätssicherung – des vorliegenden Dokumentes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datum | Geprüfte Version | Anmerkungen | Prüfer | Neuer Produktzustand |
|  |  | **RALL ADRIAN BEIM GEGENLESEN** |  |  |

Inhalt

[1 Einleitung 4](#_Toc34239361)

[2 Ausgangssituation und Zielsetzung 4](#_Toc34239362)

[3 Funktionale Anforderungen 5](#_Toc34239363)

[3.1 Use-Case Übersicht 5](#_Toc34239364)

[3.2 Use-Beschreibungen 5](#_Toc34239365)

[3.3 (Sonstige) Funktionalität 7](#_Toc34239366)

[3.4 Modell des Problembereichs (Konzeptionelles Datenmodell) 7](#_Toc34239367)

[4 Nicht-Funktionale Anforderungen 7](#_Toc34239368)

[4.1 Benutzbarkeit (Usability) 7](#_Toc34239369)

[4.2 Zuverlässigkeit (Reliability) 8](#_Toc34239370)

[4.3 Leistung (Performance) 8](#_Toc34239371)

[4.4 Unterstützbarkeit (Supportability) 9](#_Toc34239372)

[4.5 Sonstige Einschränkungen 9](#_Toc34239373)

[5 Risikoakzeptanz 10](#_Toc34239374)

[6 Skizze der Gesamtsystemarchitektur 10](#_Toc34239375)

[7 Lieferumfang 11](#_Toc34239376)

[8 Abnahmekriterien 11](#_Toc34239377)

[9 Glossar 11](#_Toc34239378)

[10 Abkürzungsverzeichnis 12](#_Toc34239379)

[11 Literaturverzeichnis 12](#_Toc34239380)

[12 Abbildungsverzeichnis 12](#_Toc34239381)

# Einleitung

Dieses Dokument enthält alle an das zu entwickelnde System gestellten Anforderungen. Die Gliederung orientiert sich am Aufbau des V-Modell-XT®[[1]](#footnote-1)-Produkts „Anforderungen (Lastenheft)“, ist jedoch zur Verwendung für die Veranstaltung **„Software-Projekte“** in Informatik-Curricula der **OTH-Amberg-Weiden** angepasst worden (und nicht konform zum V-Modell-XT): Teilnehmer dieser Veranstaltung erhalten von ihrem „Auftraggeber“ lediglich einen Überblick über das gewünschte System, was ungefähr dem Thema „Ausgangssituation und Zielsetzung“ in diesem Dokument entspricht; die Anforderungen müssen die Teilnehmer dann in enger Abstimmung mit ihrem „Auftraggeber“ selbst erarbeiten und in diesem Dokument niederlegen. Dadurch sollen sie Gelegenheit erhalten, auch Tätigkeiten der System-Analyse intensiver zu üben. Die „Auftraggeberseite“ liefert also nicht – wie im V-Modell-XT vorgesehen - das komplette Lastenheft, aus dem die „Auftragnehmerseite“ ein separates Pflichtenheft ableitet; stattdessen wird das hier vorliegende Dokument vom studentischen Entwicklerteam zur Dokumentation der Analyse-Ergebnisse erstellt und zugleich als Ersatz für die im V-Modell-XT vorgesehenen Dokumente Lasten- und Pflichtenheft verwendet.

Kern dieses Dokuments sind die funktionalen und nicht-funktionalen Anforderungen an das System, sowie eine Skizze des Gesamtsystementwurfs. Der Entwurf berücksichtigt die zukünftige Umgebung und Infrastruktur, in der das System später betrieben wird, und gibt Richtlinien für Technologieentscheidungen. Ebenfalls Teil der Anforderungen ist die Festlegung von Lieferbedingungen und Abnahmekriterien.

Die funktionalen und nicht-funktionalen Anforderungen dienen nicht nur als Vorgaben für die Entwicklung, sondern sind zusätzlich Grundlage der Anforderungsverfolgung und des Änderungsmanagements. Die Anforderungen sollten so aufbereitet sein, dass die Verfolgbarkeit (Traceability) sowie ein geeignetes Änderungsmanagement für den gesamten Lebenszyklus eines Systems möglich sind.

Im Allgemeinen sollten keine technischen Lösungen vorgegeben werden, um Architekten und Entwickler bei der Suche nach optimalen technischen Lösungen nicht einzuschränken.

# Ausgangssituation und Zielsetzung

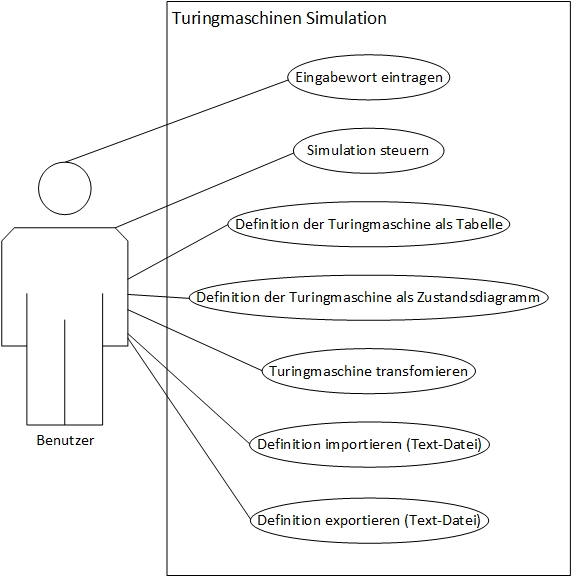
Viele im Internet auffindbare Webseiten zur eigenen Programmierung und Simulation von Turingmaschinen erlauben entweder nur die Auswahl vorgefertigter Programme mit denen eigene Eingaben überprüft werden können oder die eigene Programmierung wird nur durch viele Zeilen komplizierter Syntax möglich. Keine dieser beiden Varianten stellt Benutzer zufrieden, die sich erst in das Thema einarbeiten wollen und durch die Syntax abgeschreckt werden oder Benutzer mit Vorwissen zu Turingmaschinen, die durch die Syntax ausgebremst werden.

Das Ziel dieses Projektes ist es daher eine Lernsoftware zu erstellen, mit der man Turingmaschinen verstehen und selbst programmieren kann, von einfachen hin zu sehr komplexen. Hierfür soll der Benutzer – durch das System unterstützt – lernen, wie Turingmaschinen funktionieren, um dann immer komplexere Varianten selbst zu schreiben, ohne das komplizierte Syntax in seinen Weg gerät.

Die wichtigsten Programmfunktionen sind:

* Der Benutzer kann eine Turingmaschine in das Programm einlesen oder selbst eine erstellen und diese abspeichern, um sie später wieder zu benutzen.
* Der Benutzer kann sich die Turingmaschine als Tabelle oder als Zustandsdiagramm darstellen lassen, wobei eine freie Wahl zwischen den beiden Darstellungsarten jederzeit möglich ist.
* Das Programm stellt sicher, dass die Turingmaschine gültig ist und weist den Benutzer bei Ungültigkeit auf die Fehlerquelle(n) hin (z.B. im aktuellen Zustand existieren mehrere Übergangsregeln für ein gleiches Symbol)
* Der Benutzer hat ein endlos langes Band vor sich, welches in einzelne Felder unterteilt ist. Auf das Band wird das vom Benutzer eingetragene Eingabewort übernommen, falls alle Symbole des Eingabeworts im Eingabealphabet der Turingmaschine vorhanden sind.
* Das Programm erlaubt es dem Benutzer zur besseren Nachvollziehbarkeit des Programmablaufs diesen in geeigneter Weise anzupassen. Hierfür soll es folgende Möglichkeiten geben:
  + Simulationsdurchlauf in Einzelschritten
  + Abspielen in auswählbarer Geschwindigkeit
  + Sprung zum Programmanfang (Abbruch der Simulation)
  + Pausieren und Fortsetzen des Ablaufs
  + Aktuellen Zustandsübergang in Tabelle und Zustandsdiagramm markieren
* Das Programm informiert den Benutzer am Ende des Programmdurchlaufs darüber, ob die Turingmaschine in einem akzeptierenden Zustand gestoppt ist oder nicht.

## Use-Case Übersicht



## Use-Beschreibungen

|  |  |  |
| --- | --- | --- |
| **Eingabewort eintragen** | | |
| **Kennung** | | UC-1 |
| **Priorität** | | Hoch |
| **Kurzbeschreibung:** | | |
| Der Benutzer trägt ein Eingabewort auf der Benutzeroberfläche ein. Das System überprüft, ob alle Symbole des Eingabeworts im Eingabealphabet enthalten sind. | | |
| **Vorbedingung(en):** | | |
| * Es liegt eine vollständig definierte Turingmaschine vor. * Die Simulation ist beendet. | | |
| **Nachbedingung(en):** | | |
| * Auf dem Band der Turingmaschine befindet sich ein gültiges Eingabewort. | | |
| **Normaler Ablauf:** | | |
|  | 1. Dieser Anwendungsfall beginnt, wenn der Benutzer das Eingabewort einträgt. 2. Der Benutzer bestätigt seine Eingabe. 3. Das System überprüft anhand des Eingabealphabets das Eingabewort auf Gültigkeit. 4. Das System schreibt das Eingabewort auf das Band.   **Ende**. | |
| **Ablauf-Varianten:** | | |
| 3a | Ungültiges Eingabewort | |
|  | 1. Das System signalisiert, dass das Eingabewort ungültig ist.   **Ende.** | |
| **Spezielle Anforderungen:** | | |
|  | | |
| **Zu klärende Punkte:** | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| **Simulation steuern** | | |
| **Kennung** | | UC-2 |
| **Priorität** | | Hoch |
| **Kurzbeschreibung:** | | |
| Dem Benutzer stehen zur Steuerung der Simulation verschiedene Möglichkeiten zur Verfügung.  Dabei kann der Benutzer die:   * Simulation starten. * Simulation abbrechen. * Simulation pausieren. * Simulation fortsetzen. * Simulationsverzögerung einstellen. * Simulation in Einzelschritte durchlaufen. * Option Zustandsübergänge markieren an-/abwählen. | | |
| **Vorbedingung(en):** | | |
| * Die Definition der Turingmaschine ist gültig * Auf dem Band der Turingmaschine befindet sich ein gültiges Eingabewort. | | |
| **Nachbedingung(en):** | | |
|  | | |
| **Normaler Ablauf:** | | |
|  | 1. Dieser Anwendungsfall beginnt, wenn der Benutzer ein Befehl zur Steuerung der Simulation sendet.   **Ende**. | |
| **Ablauf-Varianten:** | | |
| 1a | Befehl **Starten** | |
|  | 1. Das System startet die Simulation im Automatischen Ablauf.   *Bem.: Automatischer Ablauf bedeutet Einzelschrittausführung mit eingestellter Verzögerung, bis das Ende der TM erreicht ist.* | |
| 1b | Befehl **Abbruch** | |
|  | 1. Das System beendet die Simulation.   *Bem.: Der aktuelle Zustand der Simulation wird wieder auf den Startzustand zurückgesetzt.* | |
| 1c | Befehl **Pausieren** | |
|  | 1. Das System pausiert die Simulation an der aktuellen Stelle. | |
| 1d | Befehl **Fortsetzen** | |
|  | 1. Das System setzt den automatischen Ablauf an der aktuellen Stelle fort.   *Bem.: Automatischer Ablauf wie bei Variante '1a' erläutert.* | |
| 1e | Befehl **Verzögerungsanpassung** | |
|  | 1. Das System verzögert anhand des eingestellten Werts die zeitlichen Abstände zwischen dem Wechsel zweier Zustände. | |
| 1f | Befehl **Einzelschritt** | |
|  | 1. Das System führt einen einzelnen Simulationsschritt aus. 2. Das System prüft, ob das Ende des Programmdurchlaufs der TM erreicht wurde. 3. Das System benachrichtigt den Benutzer, ob sich die TM in einem akzeptierenden Endzustand befindet. | |
| 1g | Befehl **Zustandsübergänge markieren an-/abwählen** | |
|  | 1. Das System hebt den aktuellen Zustandsübergang farblich hervor. | |
| **Spezielle Anforderungen:** | | |
|  | | |
| **Zu klärende Punkte:** | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| **Definition der Turingmaschine als Tabelle** | | |
| **Kennung** | | UC-3 |
| **Priorität** | | Hoch |
| **Kurzbeschreibung:** | | |
| Der Benutzer definiert:   * Kommentare * Zustände * Eingabealphabet * Bandalphabet * Zustandsübergänge * Zustandsbenennung   in einer Tabelle. Dabei kann entweder eine neue Tabelle erstellt oder eine bestehende bearbeitet werden. | | |
| **Vorbedingung(en):** | | |
| * Die Simulation ist beendet. | | |
| **Nachbedingung(en):** | | |
| * Die Definition der Turingmaschine ist gültig. | | |
| **Normaler Ablauf:** | | |
|  | 1. Der Anwendungsfall beginnt, wenn der Benutzer die Definition über Tabellenansicht auswählt. 2. Der Benutzer hat die Möglichkeit die Tabelle zu bearbeiten. 3. Der Benutzer bestätigt seine Eingabe. 4. Das System prüft, ob die Tabelle gültig ist. 5. Das System übernimmt die Tabelle als Definition der Turingmaschine.   **Ende.** | |
| **Ablauf-Varianten:** | | |
| 2a | Der Benutzer fügt eine neue Spalte (Zeichen) hinzu. | |
|  | 1. Das System fügt eine leere Spalte hinzu.   **Rückkehr nach**: 2 | |
| 2b | Der Benutzer löscht eine existierende Spalte (Zeichen). | |
|  | 1. Das System löscht die markierte Spalte.   **Rückkehr nach**: 2 | |
| 2c | Der Benutzer fügt eine neue Zeile (Zustand) hinzu. | |
|  | 1. Das System fügt eine leere Zeile hinzu.   **Rückkehr nach**: 2 | |
| 2d | Der Benutzer löscht eine existierende Zeile (Zustand). | |
|  | 1. Das System löscht die markierte Zeile.   **Rückkehr nach**: 2 | |
| 2e | Der Benutzer füllt die Tabelle aus. | |
|  | 1. Der Benutzer hat die Möglichkeit, folgende Eigenschaften festzulegen:  * Kommentare * Startzustand * Akzeptierende Zustände * Zwischenzustände * Eingabealphabet * Bandalphabet * Zustandsübergänge * Zustandsbenennung   **Rückkehr nach**: 2 | |
| 4a | Ungültige Tabelle | |
|  | 1. Das System weist den Benutzer auf die gefundenen Fehler hin.   **Ende.** | |
| **Spezielle Anforderungen:** | | |
|  | | |
| **Zu klärende Punkte:** | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| **Definition der Turingmaschine als Zustandsdiagramm** | | |
| **Kennung** | | UC-4 |
| **Priorität** | | Mittel |
| **Kurzbeschreibung:** | | |
| Der Benutzer definiert:   * Kommentare * Zustände * Eingabealphabet * Bandalphabet * Zustandsübergänge * Zustandsbenennung   in einem Zustandsdiagramm. Dabei kann entweder ein neues Zustandsdiagramm erstellt oder ein bestehendes bearbeitet werden. | | |
| **Vorbedingung(en):** | | |
| * Die Simulation ist beendet. | | |
| **Nachbedingung(en):** | | |
| * Die Definition der Turingmaschine ist gültig. | | |
| **Normaler Ablauf:** | | |
|  | 1. Der Anwendungsfall beginnt, wenn der Benutzer die Definition über Zustandsdiagramm auswählt 2. Der Benutzer hat die Möglichkeit das Zustandsdiagramm zu bearbeiten. 3. Der Benutzer bestätigt seine Eingabe. 4. Das System prüft, ob das Zustandsdiagramm gültig ist 5. Das System übernimmt das Zustandsdiagramm als Definition der Turingmaschine.   **Ende**. | |
| **Ablauf-Varianten:** | | |
| 2a | Der Benutzer fügt einen neuen Knoten (Zustand) hinzu. | |
|  | 1. Das System fügt einen neuen Knoten hinzu.   *Bem.: Der Knoten erhält hier automatisch eine Benennung, welche später bearbeitet werden kann.*  **Rückkehr nach**: 2 | |
| 2b | Der Benutzer löscht einen existierenden Knoten (Zustand). | |
|  | 1. Das System löscht den markierten Knoten. 2. Das System löscht alle ein- und ausgehenden Pfeile.   **Rückkehr nach**: 2 | |
| 2c | Der Benutzer bearbeitet einen existierenden Knoten (Zustand). | |
|  | 1. Der Benutzer hat die Möglichkeit, folgende Eigenschaften festzulegen:  * Startzustand * Akzeptierender Zustand * Zwischenzustände * Zustandsbenennung   **Rückkehr nach**: 2 | |
| 2d | Der Benutzer fügt einen neuen Pfeil (Zustandsübergang) hinzu. | |
|  | 1. Der Benutzer wählt einen Start- und Zielknoten aus. 2. Das System fügt einen neuen Pfeil zwischen den gewählten Knoten hinzu.   *Bem.: Start- und Zielknoten können identisch sein.*  **Rückkehr nach**: 2 | |
| 2e | Der Benutzer löscht einen existierenden Pfeil (Zustandsübergang). | |
|  | 1. Das System löscht den markierten Pfeil.   **Rückkehr nach**: 2 | |
| 2f | Der Benutzer bearbeitet einen existierenden Pfeil (Zustandsübergang). | |
|  | 1. Der Benutzer hat die Möglichkeit, folgende Eigenschaften festzulegen:  * Kommentar * Eingabesymbol (gelesenes Zeichen) * LSK Bewegung (links oder rechts)   **Rückkehr nach**: 2 | |
| 2g | Der Benutzer bestimmt das Eingabe- und Bandalphabet. | |
|  | **Rückkehr nach**: 2 | |
| 4a | Ungültiges Zustandsdiagramm | |
|  | 1. Das System weist den Benutzer auf die gefundenen Fehler hin.   **Rückkehr nach:** 2 | |
| **Spezielle Anforderungen:** | | |
|  | | |
| **Zu klärende Punkte:** | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| **Turingmaschine transformieren** | | |
| **Kennung** | | UC-5 |
| **Priorität** | | Mittel |
| **Kurzbeschreibung:** | | |
| Der Benutzer hat die Möglichkeit eine Turingmaschine in eine äquivalente Turingmaschine zu transformieren.  Dem Benutzer werden hierfür verschiedene Transformationsmöglichkeiten angeboten, welche er auswählen und durchführen kann:   1. Kein Übergang führt in den Startzustand und keiner der Übergänge beginnt in einem akzeptierenden Zustand. 2. Es gibt keine Zustandsübergänge mit unbewegtem Lese-/Schreibkopf. 3. Es wird niemals das Leerzeichen geschrieben. 4. Die Zustandsmenge ist in eine rechte und linke Hälfte geteilt. 5. Es gibt genau einen akzeptierenden Zustand. | | |
| **Vorbedingung(en):** | | |
| * Es liegt eine vollständig definierte Turingmaschine vor. * Die Simulation ist beendet. | | |
| **Nachbedingung(en):** | | |
| * Die transformierte Turingmaschine ist gültig. * Die transformierte Turingmaschine akzeptiert die gleiche Sprache. | | |
| **Normaler Ablauf:** | | |
|  | 1. Dieser Anwendungsfall beginnt, wenn der Benutzer eine Transformation auswählt. 2. Das System prüft die Bedingung der Transformation gemäß untenstehender Tabelle. 3. Das System führt die Transformation aus.   **Ende**. | |
| **Ablauf-Varianten:** | | |
| 2a | Die Bedingung für die ausgewählte Transformation ist nicht erfüllt | |
|  | 1. Das System gibt dem Benutzer eine entsprechende Rückmeldung.   **Ende.** | |
| 3a | Transformation 1 wurde ausgewählt | |
|  | 1. Das System stellt sicher, dass in der Definition der TM kein Zustandsübergang hinterlegt ist, dessen Zielzustand der Startzustand ist. 2. Das System stellt sicher, dass kein Übergang beginnend von einem akzeptierenden Zustand existiert.   **Ende.** | |
| 3b | Transformation 2 wurde ausgewählt | |
|  | 1. Das System überprüft, ob es Zustandsübergänge gibt, bei dem der LSK stehen bleibt.   **Ende.** | |
| 3c | Transformation 3 wurde ausgewählt | |
|  | 1. Der Benutzer wählt ein Zeichen, welches das Leerzeichen ersetzen soll. 2. Das System überprüft das Leerzeichen auf Gültigkeit 3. Das System ersetzt bei allen entsprechenden Zustandsübergängen das Leerzeichen durch das gewählte Zeichen.   **Ende.** | |
| 3d | Transformation 4 wurde ausgewählt | |
|  | 1. Das System teilt die Zustandsmenge in eine linke und eine rechte Teilmenge auf. In der linken Teilmenge befinden sich die Zustände, welche den LSK nach links bewegen. In der rechten Teilmenge befinden sich die Zustände, welche den LSK nach rechts bewegen.   **Ende.** | |
| 3e | Transformation 5 wurde ausgewählt | |
|  | 1. Das System stellt sicher, dass es genau einen akzeptierenden Zustand gibt.   **Ende.** | |
| **Bedingungen für die Zulässigkeit der einzelnen Transformationen**   |  |  | | --- | --- | | **Kommando** | **Bedingung** | | Transformation 1 | --- | | Transformation 2 | --- | | Transformation 3 | --- | | Transformation 4 | - Der Startzustand ist kein Endzustand (Transformation 1)  - LSK ist niemals unbewegt (Transformation 2) | | Transformation 5 | - Es gibt mindestens einen akzeptierenden Zustand | | | |
| **Spezielle Anforderungen:** | | |
|  | | |
| **Zu klärende Punkte:** | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| **Definition importieren (Text-Datei)** | | |
| **Kennung** | | UC-6 |
| **Priorität** | | Mittel |
| **Kurzbeschreibung:** | | |
| Der Benutzer lädt eine Definition für eine Turingmaschine in das System. Das System prüft, ob diese Definition gültig ist. | | |
| **Vorbedingung(en):** | | |
| * Die Simulation ist beendet. | | |
| **Nachbedingung(en):** | | |
| * Die Definition der Turingmaschine ist gültig. | | |
| **Normaler Ablauf:** | | |
|  | 1. Dieser Anwendungsfall beginnt, wenn der Benutzer eine Text-Datei zum Importieren auswählt. 2. Das System überprüft, ob die Text-Datei eine syntaktisch gültige Definition einer Turingmaschine darstellt. 3. Das System erstellt aus der Definition in der Text-Datei eine Turingmaschine.   **Ende**. | |
| **Ablauf-Varianten:** | | |
| 2a | Die Text-Datei enthält eine ungültige Definition | |
|  | 1. Das System weist dem Benutzer auf die ungültige Text-Datei hin.   **Ende.** | |
| **Spezielle Anforderungen:** | | |
|  | | |
| **Zu klärende Punkte:** | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| **Definition exportieren (Text-Datei)** | | |
| **Kennung** | | UC-7 |
| **Priorität** | | Niedrig |
| **Kurzbeschreibung:** | | |
| Der Benutzer gibt den Befehl die aktuelle Definition der Turingmaschine zu exportieren. Das System speichert diese in menschenlesbarer Form als Text-Datei an dem gewünschten Ablageort. | | |
| **Vorbedingung(en):** | | |
| * Die Definition der Turingmaschine ist gültig. | | |
| **Nachbedingung(en):** | | |
|  | | |
| **Normaler Ablauf:** | | |
|  | 1. Dieser Anwendungsfall beginnt, wenn der Benutzer den Befehl zum Exportieren sendet. 2. Das System fordert den Benutzer auf, einen Speicherort und einen Dateinamen zu wählen. 3. Der Benutzer wählt einen Speicherort und Dateinamen. 4. Das System exportiert die Definition der Turingmaschine in menschenlesbarer Form als Text-Datei.   **Ende**. | |
| **Ablauf-Varianten:** | | |
| 4a | Speichervorgang kann nicht ausgeführt werden (z.B. Nicht genügend Speicher vorhanden, keine Berechtigung) | |
|  | 1. Das System weist den Benutzer auf das Problem hin.   **Ende.** | |
| **Spezielle Anforderungen:** | | |
|  | | |
| **Zu klärende Punkte:** | | |
|  | | |

(Sonstige) Funktionalität

|  |  |  |
| --- | --- | --- |
| **ID** | **Beschreibung** | **Querverweise** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Modell des Problembereichs (Konzeptionelles Datenmodell)

Hier wird ein Modell des Problembereichs („konzeptuelles Datenmodell“) in Form eines oder mehrerer UML-Klassendiagramme eingefügt. Das konzeptuelle Datenmodell wird oft nicht streng getrennt von den Anforderungen erarbeitet, weil ein genaues Verständnis z.B. der Use-Cases nicht ohne Verständnis der „Konzepte“ aus dem Problembereich möglich ist. Aus diesem Grund wird das konzeptionelle Modell an dieser Stelle zusammen mit den Use-Cases dokumentiert.

# Nicht-Funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben Anforderungen an das System, die nicht-fachlicher Natur sind, jedoch entscheidend zur Anwendbarkeit des Systems beitragen. Sie definieren beispielsweise Qualitätsanforderungen, Sicherheitsanforderungen oder Performanceanforderungen.

Nicht-funktionale Anforderungen definieren grundlegende Eigenschaften eines Systems, die im Architekturentwurf berücksichtigt werden müssen. Sie können zur Abschätzung der Entwicklungskosten herangezogen werden und sollten, soweit möglich, messbar beschrieben sein.

Zur einfachen Strukturierung der Anforderungen werden diejenigen Anforderungen, die nicht eindeutig zu den funktionalen Anforderungen gehören, den nicht-funktionalen Anforderungen zugeordnet.

Die hier verwendete Einteilung unterscheidet verschiedene Arten von Anforderungen nach dem „FURPS“-Schema (**F**unctionality, **U**sability, **R**eliability, **P**erformance, **S**upportability), das auf Hewlett-Packard zurückgeht. Das FURPS-Schema ist hier noch um die Kategorie „Sonstige Einschränkungen“ erweitert. Anforderungen zur Funktionalität sind bereits im Kapitel 3 dokumentiert, in diesem Kapitel folgen lediglich alle restlichen Anforderungen.

Bei Bedarf kann dieses Schema zur Einteilung der Anforderungen auch durch ein anderes Schema (z.B. nach DIN ISO) ersetzt werden – wichtig ist nur, dass bei der Erfassung der Anforderungen überhaupt ein erprobtes Schema verwendet wird. Dies soll eine bessere Übersicht bieten und dazu beitragen, dass keine Anforderungen vergessen werden.

## Benutzbarkeit (Usability)

|  |  |  |
| --- | --- | --- |
| **ID** | **Beschreibung** | **Querverweise** |
| UR-001 | Das System soll über eine grafische Benutzerschnittstelle bedienbar sein. |  |
| UR-002 | Die Simulation kann über die Tastatur gesteuert werden. |  |
| UR-003 | Die exportierte Definition der Turingmaschine sind in menschenlesbarer Form und können im Texteditor bearbeitet werden. |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Zuverlässigkeit (Reliability)

Hier werden Anforderungen erfasst, die die Zuverlässigkeit („reliability“ = Zuverlässigkeit) des Systems betreffen. Hierunter fallen insbesondere Anforderungen an die Wiederherstellbarkeit („recoverability“) und die Verfügbarkeit („availabilty“) des Systems. Die Wiederherstellbarkeit betrifft die Fähigkeit, bei Ausfall oder Störung das Leistungsniveau wieder zu erreichen und betroffene Daten wieder zu gewinnen. Eine Kenngröße in diesem Zusammenhang ist MTTR („**m**ean **t**ime **t**o **r**epair“), die die mittlere Zeit bis zur Wiederinbetriebnahme des Systems bei einem Ausfall angibt. Die Verfügbarkeit kann als Verhältnis zwischen der Zeit, in der das System funktionsfähig ist, und der Gesamtzeit angegeben werden:

V=MTBF / (MTBF+MTTR)

MTTR: mean time to repair (s.o.)

MTBF: **m**ean **t**ime **b**etween **f**ailures (also die mittlere Zeit zwischen zwei Ausfällen)

|  |  |  |
| --- | --- | --- |
| **ID** | **Beschreibung** | **Querverweise** |
| RR-001 | Das System speichert im Falle eines Absturzes die aktuelle Turingmaschine in einer Text-Datei ab. |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Leistung (Performance)

Hierunter fallen Anforderungen an die Leistung („performance“) des Systems. Die Anforderungen beziehen sich insbesondere auf das Zeitverhalten (Ausführungsgeschwindigkeit, Antwortzeiten, Durchsatz) sowie auf das Verbrauchsverhalten (Anzahl der belegten Betriebsmittel und Dauer der Betriebsmittelbelegung).

|  |  |  |
| --- | --- | --- |
| **ID** | **Beschreibung** | **Querverweise** |
| PR-001 | ~~Das System soll nicht länger als eine Sekunde benötigen, um auf Benutzereingaben zu reagieren.~~ |  |
| PR-002 | ~~Das System soll nicht länger als fünf Sekunden benötigen, um die Definition einer Turingmaschine einzulesen oder abzuspeichern.~~ |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Unterstützbarkeit (Supportability)

Hierunter fallen Anforderungen, die Bereiche wie Anpassbarkeit („adaptability“), Testbarkeit („testability“), Wartbarkeit („maintainability“), Erweiterbarkeit („extensibility“), Lokalisierbarkeit („localizability“ = Anpassbarkeit an verschiedene Sprach- und Kulturräume) betreffen.

|  |  |  |
| --- | --- | --- |
| **ID** | **Beschreibung** | **Querverweise** |
| SR-001 | Die Benutzeroberfläche unterstützt Mehrsprachigkeit. |  |
| SR-002 | Das System soll aus Komponenten zusammengesetzt sein, die eine geringe Kopplung untereinander aufweisen und auf eine Aufgabe spezialisiert sind. |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Sonstige Einschränkungen

In manchen Fällen können von vorneherein Einschränkungen („constraints“) für Entwurf, Implementierung, Schnittstellen und Hardware des geplanten Systems bestehen, die ebenfalls als Anforderungen zu berücksichtigen sind und das bisherige „FURPS“-Schema zu „FURPS+“ (vgl. [Lar], S. 88) erweitern.

### Schnittstellen

### Implementierung

* Verwendbare Programmiersprachen:
* C#
* Java
* Zielumgebung:
* Visual Studio ~~Code~~ 2019
* Eclipse

### Entwurf

|  |  |  |
| --- | --- | --- |
| **ID** | **Beschreibung** | **Querverweise** |
| DR-001 | Der Entwurf soll den Grundsatz von der Trennung zwischen Model und View gemäß dem „Model-View-ViewModel“-Entwurfsmuster strikt einhalten. | UR-001 |
| DR-002 | Die Systemarchitektur soll so gestaltet werden, dass das System leicht um weitere Bänder ergänzt werden kann. |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Risikoakzeptanz

Für sicherheitskritische Systeme werden in diesem Thema Vorgaben für die Behandlung der Systemsicherheit festgelegt. Es wird aufgezeigt, welche Risiken im Rahmen des Systembetriebs bestehen, welche Schäden, oder auch welche Klassen von Schäden, mit welcher Wahrscheinlichkeit auftreten können und inwieweit das Eintreten eines Schadensfalls toleriert wird bzw. nicht mehr akzeptabel ist.

Die Risikoakzeptanz für die identifizierten möglichen Schadensfälle wird beispielsweise in Form einer Risikoakzeptanzmatrix dokumentiert. Die Matrix ist eine Vorgabe des Auftraggebers, in der er festlegt, bei welcher Schadensklasse und welcher Eintrittswahrscheinlichkeit er welche Risikoklasse akzeptiert.

Bei vielen Aufgabenstellungen in der Veranstaltung „Software-Projekte“ kann dieses Thema weggelassen werden.

~~Beim Abspeichern der Turingmaschine als Text-Datei ist es Aufgabe des Betriebssystems den Benutzer daran zu hindern, diese an Orten auf der Festplatte abzuspeichern, für die er keine Berechtigung besitzt. Das Programm übernimmt hierfür keine eigenständige Kontrolle und die Entwickler sind nicht für mögliche Folgeschäden belangbar, die durch das Abspeichern der Text-Datei an nicht erlaubten Orten entstehen. Weiterhin wird durch das Programm bei der Benennung der Text-Datei beim Abspeichern nicht überprüft, ob dieser Name möglicherweise mit wichtigen Dateien des Betriebssystems am selben Speicherort übereinstimmt und dieses hierdurch in seiner Ausführung gestört wird.~~

...entfällt hier...

# Skizze der Gesamtsystemarchitektur

Das reine Aufstellen von Anwenderanforderungen ohne Überlegungen zu möglichen Lösungsräumen birgt die große Gefahr, unrealistische Anwenderanforderungen zu definieren. Für die Einordnung, Systematisierung, Kategorisierung und auch Priorisierung von Anwenderanforderungen ist ein Koordinierungsrahmen hilfreich, um die Visualisierung der Anwenderanforderungen zu erleichtern.

Diese Aufgabe kann eine Gesamtsystemarchitektur leisten, die die Sichtweise des Anwenders repräsentiert und nicht die technische Sichtweise des Systemanalytikers beziehungsweise des Systemarchitekten. Das heißt, es ist eine funktionale Systemarchitektur mit Einbettung in die funktionalen Abläufe von Nachbarsystemen zu erstellen. Eine technische Systemarchitektur ist in dieser frühen Phase kaum möglich.

Des Weiteren sind die Besonderheiten der Einsatzumgebung des neuen Systems zu beschreiben, um vor allem die Anforderungen an die Systemsicherheit berücksichtigen zu können.

…hier Ihren Text einfügen…

# Lieferumfang

Die folgende Tabelle enthält alle Arbeitsergebnisse, die in der Veranstaltung „Software-Projekte“ zu dem vom Team zu liefernden „End-Produkt“ gehören – für die individuell von jedem Projektteilnehmer zu liefernden Ergebnisse lesen Sie bitte im Projektleitfaden bzw. im Projektkalender nach. Die Benotung erfolgt nicht nur auf Grundlage des lauffähigen Programms, sondern bezieht die Qualität der Analyse, des Entwurfs und des Systemtests mit ein.

|  |  |  |  |
| --- | --- | --- | --- |
| **Lfd. Nr.** | **Was?** | **Art des Dokuments** | **Bemerkungen** |
| **Ergebnis der System-Analyse** | | | |
| 1 | Das Dokument „Systemanalyse(XYZ)“ (also dieses Dokument) mit funktionalen, nicht-funktionalen Anforderungen und konzeptionellem Datenmodell. | * Siehe Vorlage. * Wird bei Projektbeginn mit einer Überblick gebenden Systembeschreibung an das Team ausgegeben. Das Dokument ist vom Team weiterzuführen und wieder abzugeben. | * Bitte auf Abgabetermin während des Semesters achten (s. Projektkalender). * Rechtzeitig vor Abgabe auf Qualitätssicherung achten (Review) |
| **Dokumentation des Systementwurfs** | | | |
| 2 | Das Dokument „Systementwurf(XYZ)“. | * Siehe Vorlage. | * Bitte auf Abgabetermin während des Semesters achten (s. Projektkalender) * Rechtzeitig vor Abgabe auf Qualitätssicherung achten (Review) |
| **Implementierung** | | | |
| 3 | Lauffähiger und getesteter Quellcode |  | Abgabe am Semesterende |
| **Test** | | | |
| 4 | Testspezifikation Systemtest | * Siehe Vorlage | * Endgültige Abgabe am Semesterende; zur Vorbereitung des Abnahmetests ist die Aufstellung der in den Abnahmetest einbezogenen Testfälle früher vorzulegen (Termin im Projektkalender) |

# Abnahmekriterien

In der Veranstaltung „Software-Projekte“ werden vom „Auftraggeber“ (in Absprache mit den Teilnehmern) rechtzeitig vor Semesterende Systemtestfälle ausgewählt, die das System dann am Tag der Abnahme ohne Beanstandung „überstehen“ muss.

# Glossar

|  |  |
| --- | --- |
| Begriff | Erklärung |
| Eingabealphabet | Besteht aus den einzelnen Zeichen, die vom Benutzer in die Tabelle oder das Zustandsdiagramm eingetragen wurden und dem leeren Zeichen. Es ist eine endliche Teilmenge des Bandalphabets, welche alle Zeichen, die der Benutzer eingeben kann, beinhaltet. |
| Bandalphabet | Menge der Zeichen, die eine Turingmaschine auf ihren Band verarbeitet. Darunter zählt auch das Leerzeichen der Turingmaschine. |
| Eingabewort | Die Zeichenkette, welche vom Benutzer auf das Band der Turingmaschine geschrieben werden soll. Dieses Wort besteht nur aus dem Eingabealphabet, aber ohne das leere Zeichen. |
| Gültige Definition | Die Turingmaschine enthält genau einen Startzustand, mindestens einen Endzustand und Zustandsübergänge bestehend aus:   * Mindestens einem Zielzustand * Dem neuen Zeichen, welches ggf. anstelle des alten geschrieben werden soll. * Der Richtung in die sich der LSK als nächstes bewegen soll. |
| Zielzustand | Der Zustand, welcher nach einem Zustandsübergang erreicht wird. |
|  |  |

# Abkürzungsverzeichnis

|  |  |
| --- | --- |
| Abkürzung | Erklärung |
| TM | Turingmaschine |
| LSK | Lese-/Schreibkopf |

# Literaturverzeichnis

**[Lar]** Larman Craig, *Applying UML And Patterns. An Introduction to Object-Oriented Analysis And Design,* Prentice Hall, 2nd ed., 2002

# Abbildungsverzeichnis

1. V-Modell® ist eine geschützte Marke der Bundesrepublik Deutschland. [↑](#footnote-ref-1)