Webanwendung zur Visualisierung von rettungsrelevanten Daten zur Einsatzunterstützung

Software-Entwicklungspraktikum (SEP)
Sommersemester 2023

Abnahmetestspezifikation

Auftraggeber
Technische Universität Braunschweig
Peter L. Reichertz Institut für Medizinische Informatik
Prof. Dr. Thomas Deserno
Mühlenpfordtstraße 23
38106 Braunschweig

Betreuer: Viktor Sobotta

Auftragnehmer:

Name	E-Mail-Adresse
Mohamed Wassim Chebili	m.chebili@tu-braunschweig.de
Omar Farouk Khayat	o.khayat@tu-braunschweig.de
Jonas Stepanik	j.stepanik@tu-braunschweig.de
Azhar Rahadian	a.rahadian@tu-braunschweig.de
Kacem Abdennabih	k.abdennabih@tu-braunschweig.de
Torben Oelerking	t.oelerking@tu-braunschweig.de
Qiyue Zhang	qiyue.zhang@tu-braunschweig.de

Braunschweig, 17. Mai 2023

Bearbeiterübersicht

Kapitel	Autoren	Kommentare
1	Torben Oelerking	keine
2	Torben Oelerking	keine
2.1	Qiyue Zhang	keine
2.2	Qiyue Zhang	keine
2.3	Qiyue Zhang	keine
2.4	Azhar Rahadian	keine
2.5	Azhar Rahadian	keine
3	Torben Oelerking	keine
3.1	Mohamed Wassim Chebili, Omar Khayat , Jonas Stepanik, Kacem Abdennabih	keine
3.2	Mohamed Wassim Chebili, Omar Khayat , Jonas Stepanik, Kacem Abdennabih	keine
3.3	Mohamed Wassim Chebili, Omar Khayat , Jonas Stepanik, Kacem Abdennabih	keine
4	Zusammen	keine

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im Allgemeinen ist das Testen von Software essenziell wichtig, da so die Qualität der zu entwickelnden Software gewährleistet werden kann. Das Testen der Software erfolgt bereits im Entwicklungsprozess, um Probleme frühzeitig zu erkennen, Verzögerungen und Gefährdungen des Projektes zu vermeiden und die gesetzten Anforderungen an das Produkt best möglich zu erfüllen und umzusetzen.
Insbesondere bei der zu entwickelnden Anwendung ist das Testen und der spätere reibungslose Betrieb wichtig, da bei falscher oder mangelhafter Verarbeitung, Simulation oder Darstellung der Unfalldaten falsche Eindrücke und Rückschlüsse seitens der Rettungskräfte bezüglich des Unfalls gezogen werden und entstehen könnten. Dies kann im schlimmsten Fall zur Gefährdung von Menschenleben führen. Um die Qualität der Software zu gewährleisten und entsprechend zu dokumentieren, orientiert sich die Testdokumentation an dem IEEE 829 Standard.

Die zu testende Softwareanwendung setzt sich zusammen aus der Simulation des Unfalls, der Weboberfläche und der Unfallatenverarbeitung. Serverseitig sind die Entgegennahme, die Bereitstellung und die Verarbeitung der Unfalldaten zu prüfen. Es ist zu prüfen, ob ISANs korrekt entgegengenommen und die benötigten Daten korrekt verarbeitet und gespeichert werden und ob im Fehlerfall das System entsprechend reagiert.
Des Weiteren ist zu testen, ob die Simulation des Unfalls mit den ermittelten Daten korrekt funktioniert und als Ergebnis die simulierten 3D-Fahrzeugmodelle sowie die Krafteinwirkungen auf die Insassen, sowie einige Basisdaten zu den Fahrzeugen zurückliefert und der Webanwendung bereitstellt. Die Clientanwendung ist in Bezug auf ihren definierten Funktionsumfang und ihre Nutzerfreundlichkeit zu testen. Hierbei sind die Unfallsuche, sowie die richtige Anzeige der Daten und Fahrzeuge auf der Karte zu Überprüfen. Des Weiteren muss die Webseitenlogik korrekt funktionieren.

Auf diese Weise soll mit Hilfe der Abnahmetestspezifikationen gewährleistet werden, dass die Softwareanwendung bei Fertigstellung Anforderungen an Funktionalität, Bedienbarkeit, Verständlichkeit und Zuverlässigkeit erfüllt.

2 Testplan

Der Testplan ist das zentrale Dokument der Qualitätssicherung und wird daher frühzeitig erstellt. Hier wird Umfang und Vorgehensweise der Qualitätssicherung beschrieben. Außerdem werden Testgegenstände und deren zu testenden Eigenschaften bzw. Funktionen identifiziert. Ferner werden die durchzuführenden Maßnahmen definiert.

2.1 Zu testende Komponenten

- **Simulation** <**C10**> In der Simulation werden Autounfälle mithilfe der Informationen aus der ISAN von BeamNG simuliert. Es ist zu prüfen, ob ein Unfall korrekt simuliert wird, insbesondere ob das 3D-Modell mit den entsprechenden Schäden des involvierten Fahrzeugs übertragen wird und der Webanwendung zur Verfügung gestellt wird.
- **GUI** <**C20**> Die grafische Benutzeroberfläche ermöglicht den Benutzern, über grafische Symbole mit dem Gerät zu interagieren. Daher ist es von Bedeutung zu testen, ob die Webanwendung angemessen auf die Befehle der Benutzer reagiert und die Daten auf der Benutzeroberfläche korrekt dargestellt werden. Hier ist es von Bedeutung zu testen, ob die Unfallsuche mittels ID korrekt funktioniert, das 3D-Mesh des Fahrzeugs und zusätzliche Informationen korrekt exportiert und auf der Karte bzw. Webseite dargestellt werden.
- **Daten** <**C30**> Die Daten fungieren als Informationsquelle für die Anwendung. Es wird geprüft, ob die korrekten Daten anhand der eingegebenen Unfall-ID bzw. ISAN abgefragt und ermittelt werden und ob nicht mehr benötigte Daten nach einer fest definierten Zeitspanne gelöscht werden.

2.2 Zu testende Funktionen/Merkmale

Folgende Funktionen sind zu testen:

- **F1** Auswertung einer ISAN
- **F2** Zusammengehörende Unfälle werden gruppiert

- **F3** Simulation eines Unfalls
- **F4** Visualisierung eines Unfalls auf der Karte
- **F5** Suchen eines Unfalls
- **F6** Löschen eines Unfalls
- **F7** Aufrufen der Rettungskarte

2.3 Nicht zu testende Funktionen

Die von den Tests ausgenommenen Funktionen umfassen das Server-Betriebssystem Windows 10 64-Bit sowie die Simulationsumgebung BeamNG. Des Weiteren werden auch das verwendete Angular-Framework und die Pakete der Programmiersprachen JavaScript und Python von den Tests ausgeschlossen. Da die genannten Produkte bereits etabliert und validiert auf dem Markt sind, ist eine weitere Überprüfung als entbehrlich anzusehen.

2.4 Vorgehen

In diesem Abschnitt werden die allgemeinen Vorgehensweisen für die einzelnen zu testenden Funktionen und Komponenten beschrieben.

2.4.1 Komponententest

Beim Komponententest wird alle in 2.1 genannte Komponente isoliert voneinander getestet. Dort werden die einzelnen Testfälle getestet und die Ergebnisse auch protokoliert. Dadurch kann Fehlerursachen genauer nachvollgezogen werden.

2.4.2 Integrationstest

Nach dem Erfolg des Komponententests werden die Komponenten vom jeweiligen Teil der Anwendung integriert und zusammengetestet. Ziel ist es, dass sie korrekt miteinander interagieren und die gewünschten Funktionalitäten bereitstellen. In diesem Fall wird die Komponente von der Webanwendung und der Simulation integriert und zusammengetestet.

2.4.3 Systemtest

Beim Systemtest wird das vollständige und integrierte System auf ihre Leistung, Funktionalität, und Konformität mit der Anforderungen überprüft und getestet. Hier werden die Webanwendung und die Simulation integriert und getestet, ob das gesamte System den erwarteten Spezifikationen entspricht.

2.4.4 Abnahmetest

Die gesamte Anwendung wird von dem Auftraggeber überprüft, um sicherzustellen, dass sie ihren Anforderungen, Erwartungen und den vereinbarten Kriterien basierend von der Pflichtenheft entspricht. Nach dem Erfolg von Abnahmetest folgt dann auch der rechtliche Abschluss des Vertrags und die Übergabe des Produkts.

2.5 Testumgebung

Für die Testphase des Backends wird ein Server-Rechner als Testumgebung verwendet. Dieser Schritt ist erforderlich, um sicherzustellen, dass das Backend einwandfrei funktioniert, bevor es in einer produktiven Umgebung implementiert wird.

Für das Frontend werden die Funktionalitäten in der Regel visuell getestet und dabei die vorgegebenen Developer-Tools genutzt. Diese Tools bieten Entwicklern und Testern die Möglichkeit, das Frontend interaktiv zu inspizieren, zu überwachen und zu debuggen.

3 Abnahmetest

In diesem Kapitel werden die Abnahmetests genauer spezifiziert. Mit Hilfe der Abnahmetests wird überprüft, ob die im Pflichtenheft festgelegten Anforderungen an die zu entwickelnde Software, vollständig und korrekt erfüllt wurden. Es wird festgelegt welche Anforderung mit welchem Testverfahren auf ihrer Korrektheit überprüft werden.

3.1 Zu testende Anforderungen

In diesem Abschnitt werden alle zu testenden Anforderungen aufgeführt

Nr	**Anforderung**	**Testfälle**	**Kommentar**
1	Funktion <F1>	<T1>, <T2>	Wird die ISAN korrekt ausgewertet?
2	Funktion <F2>	<T3>, <T4>	Werden zusammengehörende Unfälle gruppiert?
3	Funktion <F3>	<T5>	Werden Unfälle korrekt simuliert?
4	Funktion <F4>	<T6>	Werden die Daten des Unfalls korrekt übertragen und auf der Karte angezeigt?
5	Funktion <F5>	<T7>	Kann der Nutzer auf einen Unfall suchen?
6	Funktion <F6>	<T8>	Werden zu alte Unfälle vom Server entfernt?
7	Funktion <F7>	<T9>	Wird die richtige Rettungskarte geöffnet?

3.2 Testverfahren

Die Tests werden im Black-Box-Verfahren durchgeführt, indem die Ausgabe einer Funktion mit der erwarteten Ausgabe verglichen wird. Dabei werden die Tests manuell vom Entwickler durchgeführt, indem er die Eingaben selbst eingibt. Die Ausgaben werden entweder mithilfe von Tools wie Debugger oder visuell bewertet.

3.2.1 Testskripte

Es werden keine Testskripte verwendet.

3.3 Testfälle

Hier werden die Testfälle im Detail beschrieben, einschließlich der Vorbedingungen, der einzelnen Schritte und der Pass-/Fail-Kriterien, sowie anderer relevanter Informationen. Dies ermöglicht eine genaue Beschreibung und Durchführung der Tests.

3.3.1 Testfall $\langle T1 \rangle$ - Auswerten einer ISAN (mit korrektem Format)

Ziel

Hier wird getestet, ob die Anwendung die richtigen Daten aus einer ISAN mit korrektem Format extrahiert.

Objekte/Methoden/Funktionen

Funktion <F1>

Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Anwendung die korrekten Daten aus einer gültigen ISAN extrahiert und erfolgreich einen neuen Unfall auf dem Server speichert.

Vorbedingung

Eine ISAN wurde an das System übergeben.

Einzelschritte

1. Der Server empfängt eine ISAN.
2. Der Server wertet die ISAN erfolgreich aus.
3. Das REST-API gibt eine erfolgreiche Response zurück.

Beobachtungen / Log / Umgebung

Ein neuer Unfall muss erzeugt werden

Besonderheiten

Abhängigkeiten

3.3.2 Testfall $\langle T2 \rangle$ - Auswerten einer ISAN (mit ungültigem Format)

Ziel

Hier wird getestet, ob die Anwendung eine ISAN mit falschem Format erkennt.

Objekte/Methoden/Funktionen

Funktion <F1>

Pass/Fail Kriterien

Der Test ist erfolgreich, ob die Anwendung eine ISAN mit falschem Format nicht akzeptiert bzw. entsprechend reagiert.

Vorbedingung

Eine ISAN wurde an das System übergeben.

Einzelschritte

1. Der Server empfängt eine neue ISAN.
2. Der Server kann die ISAN nicht auswerten.
3. Das REST-API gibt eine fehlgeschlagene Response zurück.

Beobachtungen / Log / Umgebung

Ein neuer Unfall darf nicht erzeugt werden

Besonderheiten

Abhängigkeiten

3.3.3 Testfall $\langle T3 \rangle$ - Gruppierung zusammengehörender Unfälle

Ziel

Hier wird getestet, ob die Anwendung mehrere ISAN's, welche zeitlich und geografisch ähnliche Informationen führen, als einen einzigen Unfall identifizieren kann.

Objekte/Methoden/Funktionen

Funktion <F2>

Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Anwendung Unfälle, die innerhalb eines bestimmten Zeitraums und räumlich nahe zueinander stattfinden, korrekt als einen einzigen Unfall erkennt. Dies stellt sicher, dass nur ein Unfall erzeugt wird, der alle betroffenen Fahrzeuge enthält.

Vorbedingung

Mindestens Ein Unfall ist auf dem Server gespeichert.

Einzelschritte

1. Der Server empfängt eine neue ISAN (dieser Unfall gehört zu mindestens einem Unfall der schon gespeichert ist).
2. Der Server wertet die ISAN aus.
3. Der Server prüft, ob dieser Unfall zu mindestens einem bereits gespeicherten Unfall gehört.

Beobachtungen / Log / Umgebung

Der Unfall wird mit einem bereits gespeicherten Unfall gruppiert.

Besonderheiten

Abhängigkeiten

3.3.1

3.3.4 Testfall $\langle T4 \rangle$ - Unterscheidung auseinandergehörender Unfälle

Ziel

In diesem Test wird überprüft, ob die Anwendung richtig erkennt, dass nicht zusammengehörende Unfälle nicht gruppiert werden sollten.

Objekte/Methoden/Funktionen

Funktion <F2>

Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Anwendung Unfälle, die sich physikalisch weit voneinander entfernt befinden oder die zeitlich weit auseinander liegen, nicht in einen neuen Unfall gruppiert.

Vorbedingung

Mindestens Ein Unfall ist auf dem Server gespeichert.

Einzelschritte

1. Der Server empfängt eine neue ISAN (dieser Unfall gehört zu keinem anderen Unfall im System).
2. Der Server wertet die ISAN aus.
3. Der Server prüft, ob dieser Unfall zu mindestens einem bereits gespeicherten Unfall gehört.

Beobachtungen / Log / Umgebung

Ein neuer Unfall muss erzeugt werden.

Besonderheiten

Abhängigkeiten

3.3.1

3.3.5 Testfall $\langle T5 \rangle$ - Automatische Simulation eines Unfalls

Ziel

In diesem Test wird überprüft, ob der Simulations-Server vollautomatisch eine Simulation durchführen und die Ergebnisse liefern kann. Die Kommunikation findet dabei nur über die REST-API zwischen Web-Server und Simulations-Server statt.

Objekte/Methoden/Funktionen

Funktion <F3>

Pass/Fail Kriterien

Der Test ist erfolgreich, wenn der Simulations-Server nach eingegangener GET-Request eine Simulation startet und nach Beendigung die Ergebnisse der Simulation zurück an den Web-Server als Response übermittelt.

Vorbedingung

Auf dem Simulations-Server wird gerade keine Simulation ausgeführt.

Einzelschritte

1. Der Web-Server fragt den Simulations-Server mittels einer GET-Request nach einer Simulation mit den übermittelten Daten.
2. Der Simulations-Server empfängt die Daten und startet eine Simulation.
3. Wenn die Simulation beendet ist, liest der Simulations-Server die Ergebnisse aus.
4. Die Ergebnisse werden dem Web-Server als Response zurückgegeben.

Beobachtungen / Log / Umgebung

Besonderheiten

Die Zeit die der Simulations-Server für die Response braucht kann variieren und sollte nicht zum Fehlschlag dieses Tests führen.

Abhängigkeiten

3.3.1

3.3.6 Testfall $\langle T6 \rangle$ - Unfall visualisieren

Ziel

In diesem Test wird überprüft, ob ein Unfall korrekt auf der Karte visualisiert wird.

Objekte/Methoden/Funktionen

Funktion <F4>

Pass/Fail Kriterien

Der Test gilt als erfolgreich, wenn die Webanwendung die Daten eines Unfalls vom Server empfangen kann und den Unfall entsprechend korrekt in der Webanwendung darstellt, wobei die angezeigten Informationen den gegebenen Daten entsprechen.

Vorbedingung

Mindestens ein Unfall, dessen ID bekannt ist, ist bereits auf dem Server gespeichert.

Einzelschritte

1. Die ID eines Unfalls wird in das Suchfeld eingegeben..
2. Die Webanwendung fordert die Daten des Unfalls und der Server schickt die zurück.
3. Die Karte wird mit dem angeforderten Unfall dargestellt.

Beobachtungen / Log / Umgebung

Besonderheiten

Abhängigkeiten

3.3.1, 3.3.5, 3.3.7

3.3.7 Testfall $\langle T7 \rangle$ - Suchen eines Unfalls

Ziel

In diesem Test wird überprüft, ob die Anwendung einen Unfall anhand seiner ID finden kann.

Objekte/Methoden/Funktionen

Funktion <F5>

Pass/Fail Kriterien

Der Test gilt als erfolgreich, wenn der Nutzer die ID eines Unfalls eingeben kann und der Server den passenden Unfall finden kann und die Daten dieses Unfalls erfolgreich an die Webanwendung übermittelt werden können.

Vorbedingung

Mindestens ein Unfall, dessen ID bekannt ist, ist bereits auf dem Server gespeichert.

Einzelschritte

1. Die ID eines Unfalls wird in das Suchfeld eingegeben..
2. Die Webanwendung fordert die Daten des Unfalls
3. Der Server sucht nach einem Unfall, dessen ID der angeforderten ID entspricht
4. Der Server schickt die Daten des Unfall zurück

Beobachtungen / Log / Umgebung

Besonderheiten

Abhängigkeiten

3.3.1, 3.3.5

3.3.8 Testfall $\langle T8 \rangle$ - Löschen eines Unfalls

Ziel

In diesem Test wird überprüft, ob die Anwendung einen Unfall nach einer vorab-definierten Zeit löscht.

Objekte/Methoden/Funktionen

Funktion <F6>

Pass/Fail Kriterien

Der Test gilt als erfolgreich, wenn alle zu löschenden Unfälle erfolgreich vom Server entfernt werden. Alle Unfälle, welche die vorab-definierte Lebenszeit noch nicht überschritten haben sollten nicht gelöscht werden.

Vorbedingung

Auf dem Server muss mindestens ein Unfall existieren, welcher gelöscht werden soll und einer, welcher nicht gelöscht werden soll.

Einzelschritte

1. Alle Unfälle werden überprüft, ob ihre Lebenszeit abgelaufen ist.
2. Wird ein solcher Unfall gefunden, für den das zutrifft, wird er gelöscht.

Beobachtungen / Log / Umgebung

Besonderheiten

Abhängigkeiten

3.3.1, 3.3.3, 3.3.4

3.3.9 Testfall $\langle T9 \rangle$ - Rettungskarte aufrufen

Ziel

Der Test überprüft, ob die Rettungskarte eines Autos ordnungsgemäß geöffnet wird.

Objekte/Methoden/Funktionen

Funktion <F7>

Pass/Fail Kriterien

Wird die richtige Rettungskarte geöffnet gilt der Test als bestanden, sonst nicht.

Vorbedingung

Es muss ein Unfall mit mindestens einem Auto auf dem Server existieren.

Einzelschritte

1. Suche den Unfall
2. Richte die Karte auf ein Auto
3. Klicke auf den Button zum Öffnen der Rettungskarte eines Autos

Beobachtungen / Log / Umgebung

Besonderheiten

Dieser Test kann nur manuell ausgeführt werden.

Abhängigkeiten

3.3.7 3.3.6 3.3.4 3.3.3 3.3.1

4 Glossar

Angular ist ein TypeScript-basiertes Front-End-Webapplikationsframework. Es wird von einer Community aus Einzelpersonen und Unternehmen, angeführt durch Google, entwickelt und als Open-Source-Software publiziert.

BeamNG ist eine physikbasierte Fahrzeug-Simulation, die es Spielern ermöglicht, realistische Crash- und Fahrerlebnisse in einer offenen Spielwelt zu erleben.

IEEE 829 Standard gilt als der bekannteste Standard für Software-Testdokumentationen. Der Standard definiert eine Menge von Testdokumenten und beschreibt deren Inhalte.

ISAN (International Standard Accident Number) Das ISAN-Projekt zielt darauf ab große Datenmengen aus diversen Quellen, wie der Notfallmedizin (EMS), der elektronischen Gesundheitsakte (EHR) und Ereignisdatenschreibern (EDR) zu sammeln und durch die Schaffung einer technischen Grundlage zu verbinden und Rettungseinsätze zu unterstützen. Die ISAN selbst enthält Unfalldaten.

Python ist eine einfach zu erlernende, interpretierte, objektorientierte Programmiersprache mit dynamischer Typisierung.

3D-Mesh ist eine dreidimensionale Struktur, die eine 3D-Objekt darstellt.