HAW HAMBURG

Bachelorarbeit

Ansgar Leonard Kock

Entwicklung einer nativen, mobilen Applikation zur elektronischen Überwachung von Atemschutzeinsätzen in der Feuerwehr

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

Ansgar Leonard Kock

Entwicklung einer nativen, mobilen Applikation zur elektronischen Überwachung von Atemschutzeinsätzen in der Feuerwehr

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Wirtschaftsinformatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Lars Hamann
Zweitgutachter: Prof. Dr. Olaf Zukunft

Eingereicht am: 27. Juni 2023

Ansgar Leonard Kock

Thema der Arbeit

Entwicklung einer nativen, mobilen Applikation zur elektronischen Überwachung von Atemschutzeinsätzen in der Feuerwehr

Stichworte

Atemschutz, Überwachung, elektronisch, Feuerwehr

Kurzzusammenfassung

Die Überwachung von Atemschutzgeräteträgern in der Feuerwehr stellt eine Gewissheit im Atemschutzeinsatz dar, dass im Falle des Kontaktabbruchs Maßnahmen ergriffen werden können um die Atemschutzgeräteträger zu retten. Aktuell kommen größtenteils im Einsatzdienst elektronsiche Tafeln, die manuell beschrieben werden, zum Einsatz. Schlechte Handschrift und falsche Angaben können hier zu gravierenden Fehlern führen. Aufgabe dieser Ausarbeitung ist es, eine native, mobile Applikation zu entwickeln, die sich stark an den Vorgaben der Dienstvorschriften orientiert. Zusätzlich werden auch mögliche Schnittstellen diskutiert, mit der man die Applikation über die Atemschutzüberwachung hinaus, z.B. für das Atemschutzberichtswesen, nutzt.

Ansgar Leonard Kock

Title of Thesis

Development of a native mobile application for the electronic monitoring of SCBA operations in the fire service

Keywords

SCBA, Monitoring, Eletronic, Firefighting

Abstract

The monitoring of SCBA wearers in the fire service is a certainty in respiratory protection operations that measures can be taken to save the respirator wearer in the event of a loss of contact. Currently, electronic boards that are written on manually are used for the most part in the emergency services. Poor handwriting and incorrect information can lead to serious errors. The task of this paper is to develop a native, mobile application that is strongly oriented towards the specifications of the service regulations. In addition, possible interfaces are discussed, with which the application can be used beyond respiratory protection monitoring, e.g. for respiratory protection reporting.

Inhaltsverzeichnis

Abbildungsverzeichnis **viii**

Tabellenverzeichnis **x**

Abkürzungen **xi**

1 Einleitung **1**
1.1 Zentrale Fragestellung und Vorgehen 1

2 Grundlagen und Stand der Technik **3**
2.1 Atemschutzüberwachung in der Feuerwehr 3
2.1.1 Datenerfassung 4
2.1.2 Erfassen des Rückzugdrucks 4
2.2 Stand der Technik 4
2.2.1 Handschriftliche Erfassung 5
2.2.2 Handschriftliche Erfassung auf Formblättern 6
2.2.3 Analoge Atemschutzüberwachungstafeln 6
2.2.4 Digitale Atemschutzüberwachungssysteme 6
2.2.5 IoT-Atemschutzüberwachungssysteme 9
2.3 Object Constraint Language (OCL) 11
2.4 A UML-based Specification Environment (USE) 12
2.5 Android App-Entwicklung und Testmethodik 12
2.5.1 Roboeletric 13
2.5.2 Android Espresso 13
2.5.3 Application Exerciser Monkey 13

3 Konzeption und Entwurf der Anwendungsdomäne **15**
3.1 Anforderungsanalyse 15
3.2 Domänenmodell 16
3.2.1 UML-Diagramm der Applikation 16

3.2.2 OCL-Constraints der Applikation 18
3.3 Prozessmodellierung der Atemschutzüberwachung 21
3.4 Auswahl der Technologie 24

4 Implementierung **26**
4.1 Architektur der Applikation 26
4.2 Umsetzung der Anforderungen 27
4.2.1 Bedienung der Applikation 28
4.2.2 Verwendete Design-Pattern 29
4.2.3 Anti-Pattern: Gottklasse 29
4.3 Integration und Schnittstellen 36
4.3.1 Datenhaltung mittels SQLite 36
4.3.2 Verwendung von Layoutressourcen 36

5 Testverfahren und Beurteilung **39**
5.1 Verwendung von JaCoCo 41
5.2 Manueller Systemtest 41
5.3 Ergebnisse und Diskussion 42

6 Zusammenfassung und Ausblick **43**
6.1 Zusammenfassung der Ergebnisse 43
6.2 Retrospektive 44
6.2.1 Auswahl der Programmiersprache 44
6.2.2 Vorgehensmodell bei der Entwicklung der Applikation 44
6.2.3 Entwurf von OCL-Constraints mittels USE 45
6.2.4 Verwendung von Design-Pattern 45
6.3 Empfehlungen für künftige Arbeiten 46

Literaturverzeichnis **48**

A OCL-Constraints **53**

B Anforderungen an die Applikation **57**
B.1 Funktionale Anforderungen 57
B.1.1 Erfassen von Einsatzinformationen 57
B.1.2 Erfassen von Trupps 57
B.1.3 Uhrzeit bei Luftdruckangabe 57
B.1.4 Ermittlung des Rückzugsolldrucks 57

B.1.5 Speichern des Systemstands . . . 58
B.1.6 Laden des Systemstands . . . 58
B.1.7 Periodische Speicherung des Einsatzstands . . . 58
B.1.8 Einsatzzeiterfassung . . . 58
B.1.9 Mitteilung bei Ein-Dritteln der Einsatzzeit . . . 58
B.1.10 Mitteilung bei Zwei-Dritteln der Einsatzzeit . . . 58
B.1.11 Visuelle Darstellung der Zeiterfassung . . . 59
B.1.12 Möglichkeit zur Beendigung der Zeiterfassung . . . 59
B.1.13 Überschreiten der maximalen Einsatzzeit . . . 59
B.2 Nicht-funktionale Anforderungen . . . 59
B.2.1 Benutzerfreundlichkeit . . . 59
B.2.2 Schutz vor Fehlbedienungen . . . 59
B.2.3 Fehlerhafte Nutzereingaben . . . 60
B.2.4 Zuverlässigkeit . . . 60

C Verwendete Design-Patterns 61
C.1 Dependency Injection . . . 61
C.2 Adapter-Pattern . . . 62

D Verwendete Testpattern 63
D.1 Arrange Act Assert . . . 63
D.2 Äquivalenzklassentest . . . 63
D.3 Manueller Systemtest . . . 64

E Codeanalyse im Hinblick auf Boilerplate-Code zwischen Java und Kotlin 67

Glossar 69
Selbstständigkeitserklärung . . . 71

Abbildungsverzeichnis

2.1 Analoge Atemschutzüberwachungstafel (Eigenbau) 6
2.2 Analoge Atemschutzüberwachungstafel der Fa. Dräger (REGIS 300) . . . 8
2.3 UML-Diagramm zwischen Arbeitgeber und Arbeitsplatz 11
2.4 Mathematische Darstellung des OCL Constraints von 2.3 12

3.1 UML der Anwendungsdomäne 17
3.2 UML-Diagramm der Anwendungsdomäne erzeugt mit USE 18
3.3 Mit USE und auf Grundlage der OCL-Constraints manipulierter gültiger Systemzustand 19
3.4 Mit USE und auf Grundlage der OCL-Constraints manipulierter ungültiger Systemzustand: Falsche Einsatzstatus, kein Countdown definiert und zweimal Truppmann ohne Truppführer 19
3.5 BPMN-Modellierung eines Einsatzablaufs 23
3.6 Absatz von Tablets weltweit in den Jahren 2011 bis 2020 nach Betriebssystem (in Millionen Stück) 25
3.7 Beliebteste Programmiersprachen weltweit laut PYPL-Index im März 2023 25

4.1 Projektstruktur der Applikation 27
4.2 Beispielhafter onClickListener, mit AlertDialog und Lambda-Ausdruck, zyklomatische Komplexität von 6. 30
4.3 Erster Entwurf eines onClickListener im Rahmen eines MVPs. Der Code weist eine zyklomatische Komplexität von 6 und ein ATFD von 5 (pag hier als seperate Zugriffsmöglichkeit) auf. 31
4.4 Korrigierter Code mit einer zyklomatischen Komplexität von 3 und einer ATFD von 2 (protected void findeHöchstenDruck()). Die Methodenaufrufe werden fachlich korrekt an der Klasse Trupp ausgeführt, die entsprechenden Methodenaufrufe an die Truppmitglieder in Subroutinen der Klasse PAGeraet weiter delegiert und gesammelt zurückgegeben. 32
4.5 Grundstruktur GUI 33

4.6 Druckabfrage im Einsatz 33
4.7 Sicherheitsmechanismus beim Druck von „ENDE“ 34
4.8 Sicherheitsmechanismus beim Druck von „NEU“ 34
4.9 Die Applikation markiert automatisch den Richtwert für den Rückzugsdruck, der für den gesamten Trupp gilt. 35
4.10 Falls ein bestimmter Schwellenwert unterschritten wird, bei dem kein Rückzug mehr möglich ist, wird die überwachende Person darauf hingewiesen und die entsprechenden Drücke markiert. 35
4.11 ER-Modell der Datenbank mit den Datentypen von SQLite 38

5.1 Testprozess während der Entwicklung 40
5.2 Android-Test-Pyramide nach [26] 41

6.1 Mögliche Schnittstellen für eine Erweiterung der Applikation 46

Tabellenverzeichnis

2.1 Vor- und Nachteile von der handschriftlichen Erfassung 5
2.2 Vor- und Nachteile von analogen Atemschutzüberwachungstafeln 7
2.3 Vor- und Nachteile von IoT-Atemschutzüberwachungssystemen 10

Abkürzungen

ATFD Access To Foreign Data.

DMO Direct Mode Operation.

FwDV Feuerwehr-Dienstvorschrift.

JVM Java Virtual Machine.

OCL Object Constraint Language.

RLST Rettungsleitstelle.

TCC Tight Class Cohesion.

TMO Trunked Mode Operation.

UML Unified Modeling Language.

USE A UML-based Specification Environment.

WMC Weighted Method Count.

1 Einleitung

Der Einsatz von Atemschutz bei Feuerwehreinsätzen stellt eine der gefährlichsten Tätigkeiten der Feuerwehr dar. Der Tod vom Brandmeister Stampe im Jahre 1996 in Köln, bei dem sich der Feuerwehrmann in einer Fangleine verhedderte und durch mangelnde Vorschriften weder eine Atemschutzüberwachung noch ein Sicherheitstrupp bereitstand, um dem verunglückten Feuerwehrmann zu helfen, verstarb dieser an der Einsatzstelle [33]. In Folge dieses Unfalls wurde der Einsatz durch eine Unfallkommission aufgearbeitet und aufgrund des Schlussberichts wurde im Jahre 2002 die Feuerwehr-Dienstvorschrift (FwDV) 7 - *Atemschutz* umfangreich erarbeitet und enthält Vorschriften unter anderem zur Atemschutzüberwachung [7].
Trotz dieser Vorschriften gibt es immer wieder Zwischenfälle, so 2005, bei der eine mangelnde Kompetenz zum Bedienen der Atemschutzüberwachung fast zum „kausalen Zusammenhang mit dem Unfall“ gestanden hätte [9].

1.1 Zentrale Fragestellung und Vorgehen

Ziel dieser Bachelorarbeit ist es eine Applikation zur elektronischen Überwachung von Atemschutzeinsätzen in der Feuerwehr zu entwickeln. Die Applikation soll dazu beitragen, die Eingabesicherheit zu erhöhen und die Zahl der Anwendungsfehler zu reduzieren. In diesem Kontext stellt sich folgende Forschungsfrage:

Inwieweit kann eine digitale Applikation die manuelle Atemschutzüberwachung bei Feuerwehreinsätzen unterstützen und welche Anforderungen müssen dafür erfüllt werden?

Dabei werden im ersten Teil der Arbeit grundlegende Fachlichkeiten der Atemschutzüberwachung erläutert. Im zweiten Teil werden die fachlichen Aspekte der Informatik thematisiert, wie beispielsweise die Object Constraint Language (OCL). Zudem werden spezifische Themen wie die Entwicklung von Android-Applikationen behandelt. Nachdem

die Grundlagen verdeutlicht wurden, wird im nächsten Schritt der Entwurf der Android-Applikation entwickelt. Hierbei werden die Anforderungen und Funktionalitäten festgelegt. Anschließend wird die Implementierung der Applikation umgesetzt. Dabei werden die zuvor festgelegten Design-Entscheidungen in Code umgesetzt. Um den Entwurf und die Entwicklung zu validieren, wird die Software im Anschluss getestet. Es werden verschiedene Testfälle entwickelt, um sicherzustellen, dass die Applikation ordnungsgemäß funktioniert und die definierten Anforderungen erfüllt. Abschließend werden die Ergebnisse der Arbeit zusammengefasst und ein Ausblick für zukünftige Arbeiten gegeben. Hierbei werden die erreichten Ziele des Projekts bewertet und mögliche Verbesserungen oder Erweiterungen für die Zukunft diskutiert.

2 Grundlagen und Stand der Technik

Im folgenden Kapitel werden die grundlegenden Prinzipien und Konzepte erläutert, die für das Verständnis und die Entwicklung der Applikation vorausgesetzt werden. Der erste Teil wird sich auf feuerwehrtechnische Grundlagen fokussieren, während in der zweiten Hälfte dieses Kapitels verschiedene Aspekte der Softwareentwicklung, die in der Arbeit Einzug finden, erläutert werden.

2.1 Atemschutzüberwachung in der Feuerwehr

Grundlage für diese Arbeit stellt die FwDV 7 - *Atemschutz* dar. Als Empfehlung ist sie in allen 16 Bundesländern der Bundesrepublik Deutschland umgesetzt [31]. Keine zugänglichen Informationen gibt es im Bezug auf die Bundeswehrfeuerwehr. Formal festgelegt ist nur, welche Informationen die Atemschutzüberwachung erfassen muss, jedoch nicht, welche Systematik zur Erfassung genutzt werden muss [7, S.21][17, S.169]. Daraus ergeben sich eine Vielzahl von Systemen, die im Abschnitt Stand der Technik (2.2) ausführlich erörtert werden. Bedingt durch die zeitlich begrenzte Einsetzbarkeit durch die Faktoren der Erschöpfung und des begrenzten Luftvorrats der umluftunabhängigen Atemschutzgeräte (i. d. R. 200-300bar) ist eine Überwachung zum einen eine sicherheitstechnische Einrichtung, zum anderen aber auch eine einsatztaktische, denn der Einheitsführer kann sich so einen Überblick über die im Einsatz befindlichen Trupps an Atemschutzgeräteträgern verschaffen [7, S.26]. Durch die Atemschutzüberwachung ist gewährleistet, dass ein regelmäßiger Funkkontakt zu den Einsatzkräften hergestellt wird. Kommt kein Kontakt zustande, so können innerhalb kürzester Zeit entsprechende Gegenmaßnahmen ergriffen werden.

2.1.1 Datenerfassung

Die FwDV 7 regelt nach [7, S.21] und [17, S. 169], welche Daten bei einem Atemschutzeinsatz grundsätzlich erfasst werden sollen:

1. Datum
2. Einsatzort
3. Namen der Einsatzkräfte des Trupps
4. Funkrufnamen des Trupps
5. Uhrzeit + Behälterdruck beim Anschließen des Atemschutzgerätes
6. Uhrzeit + Behälterdruck bei 1/3 und 2/3 der zu erwartenden Einsatzzeit
7. Uhrzeit + Behälterdruck bei Erreichen des Einsatzziels
8. Uhrzeit + Behälterdruck bei Beginn des Rückzuges

2.1.2 Erfassen des Rückzugdrucks

Gemäß FwDV 7 wird für den Rückweg grundsätzlich der doppelte Luftvorrat als Maßgabe genommen wie für den Hinweg. Daraus lässt sich ableiten, dass auch bei Nichterreichen des Zieles ab einem bestimmten Wert der Rückzug anzutreten ist. Dabei ist das Truppmitglied mit dem höchsten Luftverbrauch der Richtwert für den gesamten Trupp [30, S.33][7, S.27].

2.2 Stand der Technik

Seit der Einführung der Atemschutzüberwachung im Feuerwehreinsatz sind verschiedene Systeme entwickelt worden. Dabei legen u.a. die DIN-Normen für die Fahrzeugbeladung fest, dass ein sogenanntes *Atemschutzüberwachungssystem* Bestandteil der Beladung sein muss [18].
Genauere Anforderungen an ein solches System sind bereits unter 2.1 erläutert worden. Im Folgenden werden verschiedene *Atemschutzüberwachungssysteme* vorgestellt, die erwerbbar sind.

Vorteile	**Nachteile**
Einfachheit – die handschriftliche Erfassung ist jederzeit mit den erforderlichen Mitteln möglich.	Fehleranfälligkeit – Insbesondere in Stresssituationen
Flexibel – die Atemschutzüberwachung ist nicht ortsgebunden	Keine Weiterverarbeitung – Ist das Schriftbild der überwachenden Person zu schlecht oder unleserlich, kann dies zu Problemen führen.
Günstig – die Anschaffungskosten für Papier und Zettel sind überschaubar	Keine Echtzeitdaten – Die Daten müssen manuell übertragen werden, was zu Verzögerungen oder Fehlern führen kann.
Keine Stromversorgung – Zettel und Stift benötigen keine Stromversorgung und sind unabhängig von Stromausfällen	Keine Möglichkeiten zur automatisierten Weiterverarbeitung – Die Aufzeichnungen können nicht automatisiert weiterverarbeitet werden.
Wartung – Keine professionelle Wartung erforderlich.	Witterung – Wetter kann die Aufzeichnungen unleserlich machen, wenn sie nicht geschützt sind.

Tabelle 2.1: Vor- und Nachteile von der handschriftlichen Erfassung

2.2.1 Handschriftliche Erfassung

Um die unter 2.1.1 geforderten Daten zu erfassen genügen Stift und Papier sowie ein Zeitmesser. Prinzipiell ist der Aufwand hier aber als hoch einzustufen, da sich die überwachende Person zunächst eine Systematik überlegen muss, wie diese Daten erfasst werden sollen (Tabelle, Liste etc.). Hier ist eine hohe Fehleranfälligkeit gegeben, da hier bei der Erfassung der Daten wichtige Punkte vergessen werden könnten. Dies könnte insbesondere in Stresssituationen wie dem Einsatz passieren.

Diese Methode sollte nur noch im äußersten Notfall praktiziert werden. Insbesondere, da der Schritt zur Stufe Erfassung auf Formblättern eine ebenso einfache Methode darstellt, die schnell umsetzbar ist.

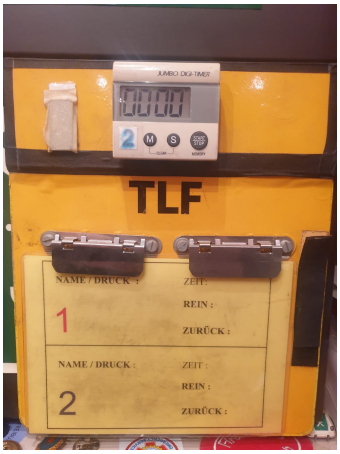


Abbildung 2.1: Analoge Atemschutzüberwachungstafel (Eigenbau)

2.2.2 Handschriftliche Erfassung auf Formblättern

Formblätter stellen in vielen Fällen auch die Grundlage für analoge Atemschutzüberwachungstafeln dar. Dabei wird eine Grundstruktur für die Erfassung der erforderlichen Daten nach 2.1.1 vorgegeben. Sie limitieren den Anwender in seinen Eingabemöglichkeiten und können nützliche Hinweise, wie zum Beispiel die Errechnung von Rückzugsdrücken oder Vorgaben zum Ausfüllen beinhalten. Unter der Abbildung 2.2 findet sich ein solches Formblatt.

2.2.3 Analoge Atemschutzüberwachungstafeln

Analoge Atemschutzüberwachungstafeln sind mechanische oder elektronische Tafeln, welche sich durch einen integrierten Zeitmesser und ein Formblatt oder einer Möglichkeit zur Dokumentation mittels Stift auszeichnen. Einige Tafeln können optische oder akustische Warnsignale von sich geben [20].

2.2.4 Digitale Atemschutzüberwachungssysteme

Digitale Atemschutzüberwachungssysteme sind Systeme, die als Endanwendungen auf mobilen Endgeräten laufen. Sie limitieren den Anwender stark in seinen Eingaben durch die Vorgabe von Eingabefeldern. Zusätzlich sorgt eine mögliche Automatisierung dafür, dass Fehler bei der Eingabe vermieden werden können (Datenvalidierung) und dass Werte

Vorteile	**Nachteile**
Geringe Kosten – im Vergleich zu IoT-Atemschutzüberwachungs-systemen oder digitalen Atemschutzüberwachungssys-temen sind die Kosten einer Tafel überschaubar.	Fehleranfälligkeit – Insbeson-dere in Stresssituationen
Flexibel – die Atemschutz-überwachung ist nicht ortsge-bunden	Keine Weiterverarbeitung – Ist das Schriftbild zu schlecht oder unleserlich, kann dies zu Problemen führen.
Feedback – Akustisches und optisches Feedback bei Errei-chen von Zeitintervallen	Keine Echtzeitdaten – Die Da-ten müssen manuell übertra-gen werden, was zu Verzöge-rungen oder Fehlern führen kann.
Genaue Zeitmessung – eine in-tegrierte Uhr misst die Ein-satzzeit genau	Keine Möglichkeiten zur au-tomatisierten Weiterverarbei-tung – Die Aufzeichnungen können nicht automatisiert weiterverarbeitet werden.
Wartung – Keine professionel-le Wartung erforderlich. (ggfs. Batteriewechsel der Uhr)	Witterung – Wetter kann die Aufzeichnungen unleser-lich machen, wenn sie nicht ge-schützt sind.

Tabelle 2.2: Vor- und Nachteile von analogen Atemschutzüberwachungstafeln

Abbildung 2.2: Analoge Atemschutzüberwachungstafel der Fa. Dräger (REGIS 300)

wie die Errechnung von Einsatzzeiten oder Rückzugsdrücken automatisiert ausgerechnet werden [15].

2.2.5 IoT-Atemschutzüberwachungssysteme

IoT-Atemschutzsysteme sind die Weiterentwicklung von digitalen Atemschutzüberwachungssystemen, bei denen die Atemschutzüberwachung durch IoT-Devices ergänzt oder vollständig übernommen wird. Mögliche Erweiterungen mit IoT-Devices sind nach [19] und [37, S.10-12]:

- Registrierung der Einsatzkraft mittels RFID-Chip
- Überwachen der Hochdruckleitung des Pressluftatmers mittels IoT-Manometer
- Überwachung von Vitalwerten wie Pulsoxymetrie, Pulsfrequenz, Körperkerntemperatur
- Totmannwarner

Problematisch ist der Einsatz, wenn die Abhängigkeit ausschließlich von IoT-Devices ausgeht. Jedes IoT-Device benötigt eine eigene Stromversorgung (i.d.R. einen Akku) und ist als Slave von einem Master abhängig, welcher meist ein Gateway darstellt [21]. Insbesondere Keller, weitläufige Einsatzstellen oder Bunkeranlagen können hier zu Empfangsproblemen zwischen den Modulen führen. Auch die Erhebung von sogenannten besonders schutzwürdigen Daten, wie Gesundheitsdaten, bedürfen softwareseitiger Sicherheitsvorkehrungen und müssen im Einklang mit den Datenschutzverordnungen stehen [37, S.32].

Vorteile	**Nachteile**
Reduzierung des Funkverkehrs – durch die Ergänzung der IoT-Devices (z.B. durch ein IoT-Manometer) müssen die Drücke nicht mehr per Funk übermittelt werden.	Kosten – Die Anschaffung eines IoT-Überwachungssystems ist im Vergleich zu den anderen Systemen deutlich teurer.
Überwachung an verschiedenen Parametern – Durch die Kombination verschiedener Metriken lässt sich der Zustand einer Einsatzkraft besser beurteilen.	Wartung – die Wartung der Systeme erfordert Fachwissen.
Schnelligkeit – durch die Registrierung mittels RFID-Chip beschleunigt sich die Einsatzbereitschaft	Cyberangriffe – verteilte Systeme mit IoT-Devices sind anfällig für Cyberangriffe. Insbesondere im Hinblick auf besonders schutzwürdige Daten ist eine Abwägung erforderlich.
Datenverarbeitung – die gesammelten Daten könnten mittels Schnittstelle weiterverarbeitet werden.	Energieversorgung – sämtliche Akteure des Systems benötigen Strom. Dies kann im schlimmsten Fall zu Ausfällen führen.
	Empfangsprobleme – größere Einsatzstellen oder in Gebäuden können dafür sorgen, dass die IoT-Devices ihre Gateways trotz Repeatereinsatz nicht mehr erreichen können.

Tabelle 2.3: Vor- und Nachteile von IoT-Atemschutzüberwachungssystemen

2.3 Object Constraint Language (OCL)

Object Constraint Language (OCL) ist eine formale Sprache, welche für die Beschreibung von Begrenzung und Bedingungen im Kontext von Unified Modeling Language (UML) verwendet wird. Sowohl die UML als auch die OCL wurde von der Object Management Group (OMG) standardisiert [12][36]. Die Sprache ermöglicht es dargestellte Objekte und Klassen formal so zu definieren, dass komplexe Bedingungen und Einschränkungen hinsichtlich der Attribute und der Erzeugung von Objekten definiert werden können. Beispielhaft kann man die Beziehung zwischen Arbeitgeber und Arbeitsplatz betrachten: Abbildung 2.3 Statisch beschreibt UML hier eine Assoziation zwischen Arbeitgeber und Arbeitsplatz. Jedoch werden dynamische Bedingungen, die zur Laufzeit gelten müssen, hier nicht festgehalten. OCL ermöglicht es nun uns festzulegen, dass mindestens ein Arbeitsplatz im Kontext von Arbeitgebern existieren muss. Ein möglicher OCL-Constraint könnte wie folgt aussehen:

```
context Arbeitgeber
inv mindestens_ein_Arbeitsplatz: self.Arbeitsplatz->size() >= 1
```

Im Kontext dieser Arbeit wird OCL verwendet, damit insbesondere Dienstvorschriften, die erfüllt werden müssen, bereits in der Anwendungsdomäne definiert werden können und im späteren Entwurfs- und Entwicklungskontext umgesetzt werden können.

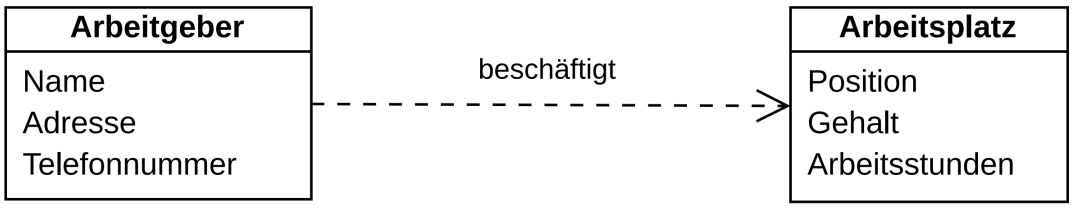


Abbildung 2.3: UML-Diagramm zwischen Arbeitgeber und Arbeitsplatz

$$\forall a \in \texttt{Arbeitgeber} \mid \texttt{a.Arbeitsplatz.size()} \geq 1 \tag{2.1}$$

Abbildung 2.4: Mathematische Darstellung des OCL Constraints von 2.3

2.4 A UML-based Specification Environment (USE)

A UML-based Specification Environment (USE) ist ein Open-Source-Tool, was im Kontext der OCL (2.3) und UML verwendet wird. Das Tool bietet u.a. die Möglichkeit OCL-Constraints zu validieren, indem es Systemzustände erstellt und manipuliert. Der Anwender gibt in einem speziellen Dateiformat ein UML-Diagramm und dazugehörige OCL-Constraints vor. Innerhalb der Anwendung kann dieser die Systemzustände manipulieren um zu prüfen, ob die OCL-Constraints die gewünschten Bedingungen erfüllen. Bereits beim Erstellen der eigenen Konfiguration prüft USE, ob die definierten Bedingungen Fehler aufweisen und gibt dem Nutzer ein Feedback, ob das Modell geladen werden konnte. Über die Funktion des Klassendiagramms lässt sich die statische Beziehung zwischen Assoziationen und Klassen anzeigen. Dabei können auf Grundlage dessen UML-Diagramme exportiert werden, so wie auch hier unter 3.2 geschehen. Im Objektdiagramm lassen sich dann Systemzustände durch Erzeugen von Objekten manipulieren. Dabei lassen sich separat die Struktur des Diagramms, bedingt durch die Assoziationen und die Klasseninvarianten bedingt durch die Definierten Invarianten der Klasse prüfen. Die Systemzustände lassen sich ähnlich wie die UML-Diagramme auch exportieren, so wie hier unter 3.3 und 3.4. Die Überprüfung von Systemzuständen kann automatisiert erfolgen, indem der Model Validator zusammen mit der Vorgabe von Mindest- und Höchstanzahl von Klasseninstanzen genutzt wird. Auf diese Weise kann effizient geprüft werden, ob ein gültiger Systemzustand erreichbar ist [23][25].

2.5 Android App-Entwicklung und Testmethodik

Während das Testen von Methoden in normalen Java-Anwendungen mit JUnit kein Problem darstellt, stehen sind bei der Verwendung der AndroidSDK eine Vielzahl von Komponenten beteiligt, die untereinander, miteinander agieren (Activities, Services, Content Providers usw.). Damit diese getestet werden, müssen diese unter einer echten Android-Plattform ausgeführt werden [2]. Ein möglicher Ansatz einen reinen JUnit-Komponenten-Test durchzuführen, ist die Verwendung von AndroidJUnitRunner. Die Verwendung die-

ses Ansatzes erfordert allerdings die Benutzung eines Emulators oder eines echten Gerätes und ist somit resourcenintensiv [1]. Eine ressourcenarme Alternative stellt das Verwenden von Shadow-Klassen dar. Diese benötigen keinen Emulator oder echtes Gerät zum Ausführen. Ein Ansatz mit Shadow-Klassen wird im nächsten Abschnitt unter *Roboeletric* erläutert. Zum Testen einzelner Methoden, die unabhängig von der AndroidSDK funktionieren, wurde während der Entwicklung JUnit eingesetzt.

2.5.1 Roboeletric

Roboeletric ist ein Framework, welches es ermöglicht Tests auszuführen ohne auf Emulatoren oder Android-Geräte zuzugreifen. Dabei führt Roboelectric die Tests direkt in der Java Virtual Machine (JVM) aus. Roboelectric setzt dabei auf sogenannte Shadow-Klassen, welche vereinfachte Implementierungen von Android-Klassen darstellen sollen. Vorteile gegenüber der Verwendung von Mockito ist, dass die Android-Vererbungshierarchie innerhalb von Roboeletric eingehalten wird und die Methoden ein ähnliches Verhalten wie das AndroidSDK aufweisen [41].

2.5.2 Android Espresso

Android Espresso ist ein Framework für automatisierte UI-Tests von Android Apps. Espresso ist von Google entwickelt worden und ist in die offizielle IDE Android Studio integriert. Espresso nutzt einen Emulator oder ein echtes Gerät um die Tests auszuführen (Instrumented Tests). Dabei kann man unter Espresso Eingaben sowie Knopfdrücke simulieren und Soll/Ist-Werte innerhalb der GUI vergleichen. Espresso bietet nur Black-Box-Tests von der GUI aus an und bietet keine Möglichkeit auf die Methodenabläufe oder auf andere Backend-Funktionalitäten zuzugreifen [3].

2.5.3 Application Exerciser Monkey

Der Application Exerciser Monkey oder auch umgangssprachlich *Monkey*, ist ein Tool, das in Rahmen des Testens von Android-Applikationen verwendet wird. Dabei simuliert der Monkey menschliche Benutzereingaben pseudozufällig. Hauptziel dieser Testmethodik ist es, die Anwendung unter verschiedenne Belastungs- und Nutzungsszenarien zu testen, um sicherzustellen, dass sie stabil, robust und fehlerfrei funktioniert. Durch die

Simulation von verschiedenen Nutzereingaben können Entwickler potentielle Abstürze, Hänger, Ressourcenlecks oder andere unerwünschte Verhaltensweisen identifizieren [4].

3 Konzeption und Entwurf der Anwendungsdomäne

Das Kapitel Konzeption und Entwurf der Anwendungsdomäne beschäftigt sich mit dem Softwareentwurf der Applikation. Dazu wird zunächst ein allgemeinverständliches Domänenmodell erstellt. Auf Grundlage der Feuerwehr-Dienstvorschriften wird das Modell mithilfe von OCL-Constraints und UML verfeinert. Dabei kommt das Tool USE zum Einsatz, welches unterstützt die OCL-Constraints zu validieren.

3.1 Anforderungsanalyse

Bereits in den Grundlagen wurde sich mit dem Atemschutzeinsatz in der Feuerwehr auseinandergesetzt. Dabei werden, wie bereits vorgestellt, durch die FwDV 7 genaue Anforderungen an die Atemschutzüberwachung gestellt. Aus diesen Anforderungen lassen sich auch die Anforderungen an die Applikation ableiten. Im Anhang B findet sich ein vollständiger Katalog von funktionalen und nicht-funktionalen Anforderungen an diese Applikation wieder. Um das Vorgehen beim Erstellen zu erläutern, werden im folgenden zwei gestellte Anforderungen an die Software und ihre Herleitung erläutert: B.1.1 Der Anwender muss grundlegende Informationen zum Einsatz, wie Datum, Einsatzort, Einheitsführer und Rufgruppe erfassen können. Ein Teil dieser Anforderungen ergeben sich aus [7, S.21]. Die Rufgruppe steht hierbei nicht in den Voraussetzungen nach FwDV. Jedoch ist es sinnvoll diese zusätzlich zu erfassen, da es eine Vielzahl an Funkkanälen im Direct Mode Operation (DMO)- und Trunked Mode Operation (TMO)-Modus der Funkgeräte gibt [38].
B.1.4 Das System muss automatisch errechnen, wann der Trupp sich spätestens nach Erreichen des Ziels zurückziehen muss. Diese Anforderung stellt keine zwingende Voraussetzung nach FwDV 7 dar. Jedoch ist die Umsetzung der Errechnung, die unter 2.1.2

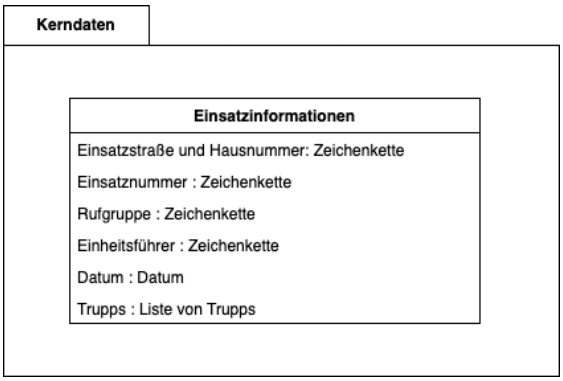
erläutert wurde, entwicklungstechnisch einfach umzusetzen und stellt so eine Erleichterung für den Anwender dar. Deshalb wurde diese in den Anforderungskatalog übernommen.

3.2 Domänenmodell

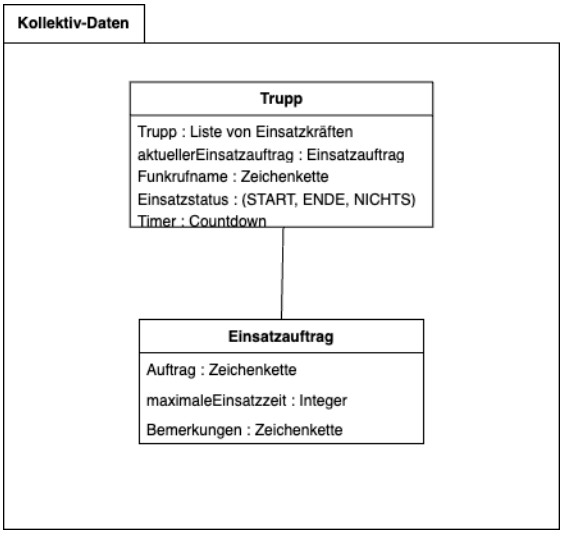
Die Domäne der Atemschutzüberwachung wird in diesem Falle in drei Segmente gegliedert. Unterschieden wird dabei zwischen sogenannten Kerndaten, die die Grunddaten des Einsatzes darstellen, sowie den Kollektivdaten, welche Informationen über die Trupps beinhalten und über die Personen-Daten, welche individuelle Informationen zu Truppmitgliedern enthalten. Berücksichtigt wurden beim Erstellen des Domänenmodells die Grundlagen, welche unter 2.1 erläutert worden sind. Die Kerndaten beinhalten die grundlegenden Einsatzinformationen des Einsatzes. Dabei hält die Einsatzinformation eine Referenz der Liste von Trupps. Von Einsatzinformationen soll es zu jedem Einsatz nur ein Exemplar geben. Von den Einsatzinformationen können wir auf die Trupps zugreifen. Die Kollektivdaten halten die Informationen der ugs. Kollektive, im fachlichen Kontext sind das die Einheiten Trupps. Ein Trupp hält seine Einsatzkräfte. Zusätzlich befiehlt ihm der Einheitsführer über den sogenannten Einsatzbefehl, daraus ergibt sich ein Einsatzauftrag. Von Kollektivdaten soll es mehrere Exemplare geben, da es im Einsatz auch mehrere Trupps und mehrere Einsatzaufträge gibt. Ein Trupp hält eine Liste von seinen Einsatzkräften. Auf die Personendaten wird über den Trupp zugegriffen. Sie enthalten individuelle Daten zu der jeweiligen Einsatzkraft. Dabei sind der Name und die einsatztechnische Qualifikation die Daten, die eine Einsatzkraft identifizieren. Zusätzlich wird jeder Einsatzkraft ein PA-Gerät zugeordnet. Dieses PA-Gerät enthält eine Liste von Schlüssel-Wert-Paaren, die den Luftdruck der Einsatzkraft zu dem jeweiligen Einsatzstand (Status) darstellt.

3.2.1 UML-Diagramm der Applikation

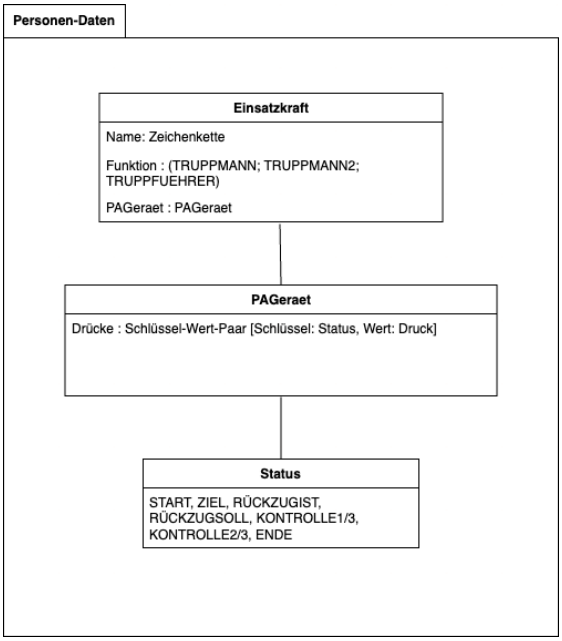
Das UML-Diagramm unter 3.2 ist im Rahmen des Entwurfes von OCL-Constraints unter Verwendung von USE entstanden. Einige Attribute des ersten Entwurfs sind dabei durch Assoziation redundant geworden. So ist die Liste von Trupps unter 3.1a durch eine entsprechende Assoziation abgelöst worden. Auch das Attribut Liste von Einsatzkräften



(a) Kerndaten (UML) der Domäne



(b) Kollektiv-Daten (UML) der Domäne



(c) Personen-Daten (UML) der Domäne

Abbildung 3.1: UML der Anwendungsdomäne

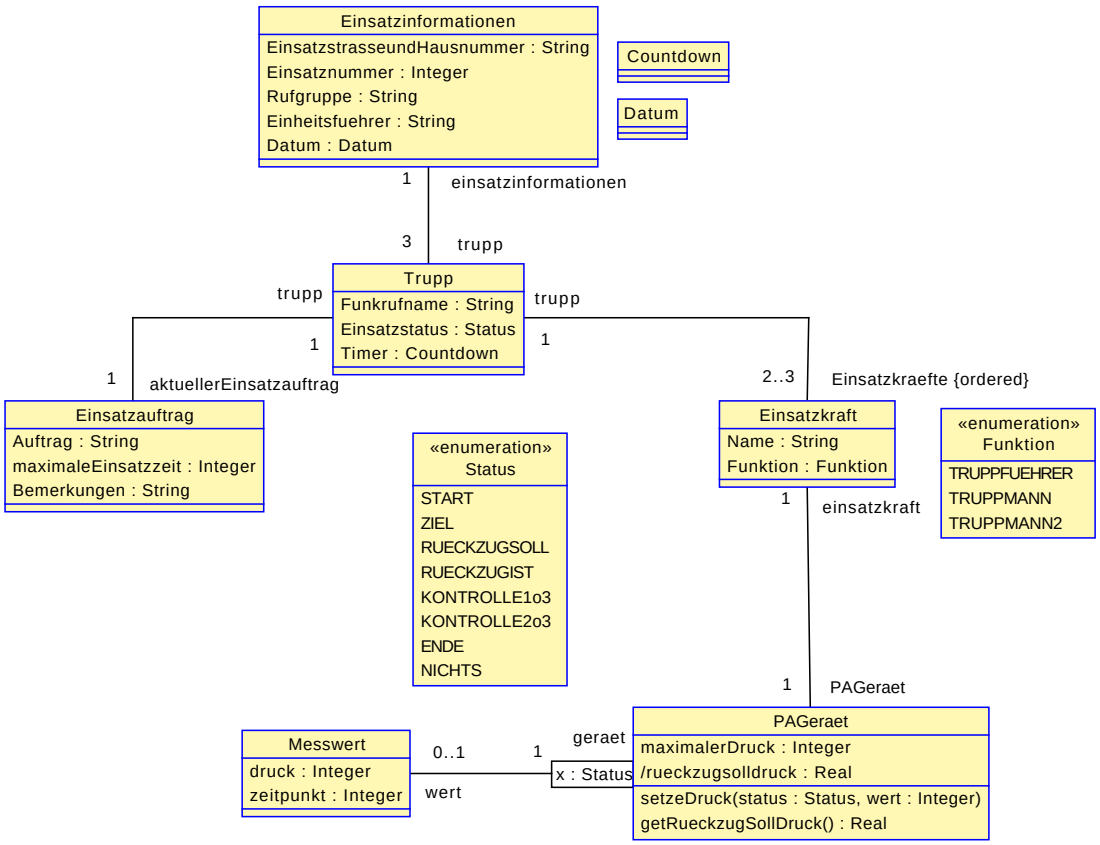


Abbildung 3.2: UML-Diagramm der Anwendungsdomäne erzeugt mit USE

und Einsatzauftrag unter Trupp 3.1b sowie die Verbindung von Einsatzkraft zu PA-Gerät 3.1c entfallen aus demselben Grund.

3.2.2 OCL-Constraints der Applikation

Ziel der Anwendung soll es nicht sein den Nutzer so stark in seinen Eingaben zu limitieren, dass er Workarounds sucht, um die Software zu nutzen. Selbst die FwDV 3 und 7 sieht vor von der FwDV abzuweichen, um einen Einsatzerfolg zu garantieren [8, S.6][7, S.32]. Hierbei muss dem Nutzer ein Handlungsspielraum eingeräumt werden, in dem die OCL-Constranints verletzt werden dürfen. Im unteren Abschnitt *Erläuterungen zu den entwickelten OCL-Constraints* wird auf diesen Punkt näher eingegangen. Um die entworfenen OCL-Constraints zusammen mit den Klassen im obigen UML-Diagramm

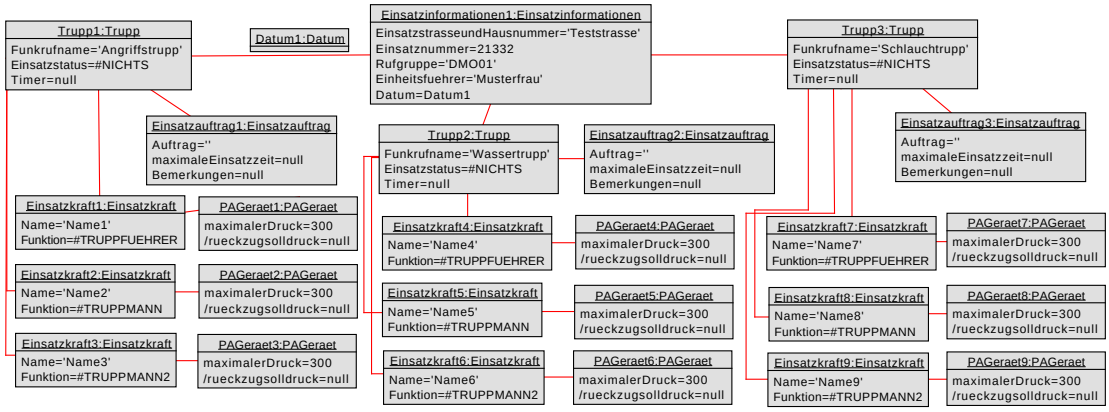


Abbildung 3.3: Mit USE und auf Grundlage der OCL-Constraints manipulierter gültiger Systemzustand

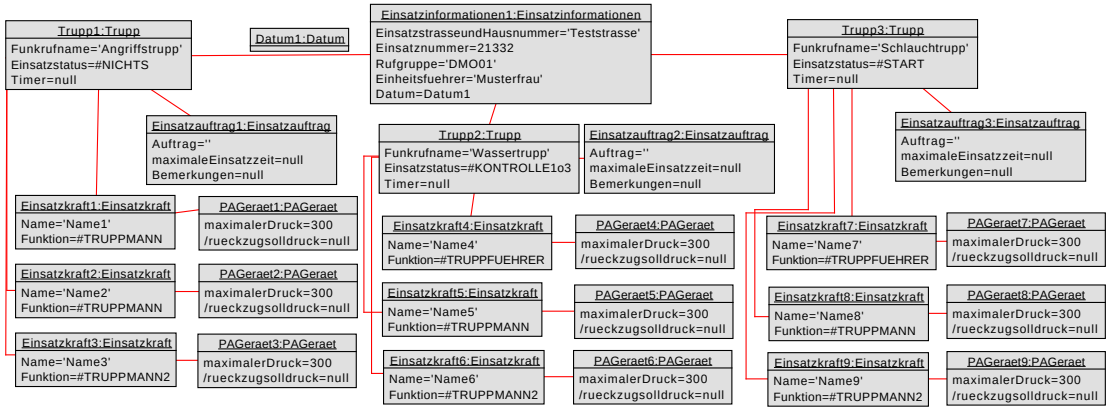


Abbildung 3.4: Mit USE und auf Grundlage der OCL-Constraints manipulierter ungültiger Systemzustand: Falsche Einsatzstatus, kein Countdown definiert und zweimal Truppmann ohne Truppführer

auf ihre Validität zu überprüfen, wurde USE verwendet. Dabei wurden während des Entwicklungsprozesses einige Constraints verworfen und andere Constraints wurden ergänzt. USE eignete sich insbesondere dazu zu verstehen, dass bestimmte OCL-Constraints auch unter anderen Bedingungen gelten und daher sinnvoll ergänzt werden müssen um den Zweck nicht zu verfehlen. Die vollständig entwickelten OCL-Constraints können dem Anhang unter A entnommen werden. Die OCL-Constraints sind einer ID zugewiesen (durch einen Kommentar dahinter). Die Stellen, an denen im Code dieser Constraint umgesetzt wurde, sind mit einem entsprechenden Kommentar versehen.

Erläuterungen zu den entwickelten OCL-Constraints

Wie im oberen Absatz erwähnt, sollen Workarounds vermieden werden. Die erarbeiteten OCL-Constraints sollen einen Kompromiss darstellen, der bestimmte Einsatzprinzipien und Flexibilität miteinander vereint. Festgelegt nach 2.1.1 ist bereits, dass Datum, Einsatzort (gegeben durch Hausnummer und Straße) definiert sein müssen. Insbesondere durch die Gegebenheit, dass im Funkverkehr sowohl im DMO-Modus als auch im TMO-Modus eine Vielzahl an Funkkanälen existieren, ist es erforderlich zu wissen, auf welchem Funkkanal die Überwachung stattfindet [35, S.4]. Dies könnte z.B. relevant werden, wenn die überwachende Person wechselt. Die Limitierung an Trupps ist bedingt durch die Empfehlung maximal drei Trupps zu überwachen [30, S. 80]. Für die Atemschutzüberwachung selber ist aber mindestens ein Trupp erforderlich. Damit ein Trupp überwacht werden kann, bedarf es zwingend einen Einsatzauftrag, dieser Umstand ist durch den unselbstständigen Trupp bedingt. Dieser Auftrag muss vor der Überwachung definiert werden. Ein Einsatzauftrag soll für jeden Trupp definiert werden, da dieses Klassenattribut auch für die Verortung genutzt werden soll. Auch wenn die Verortung, als auch der Einsatzauftrag gleich ist, so muss die Option bestehen jederzeit die Verortung ändern zu können, damit jederzeit jeder Trupp für sich lokalisiert werden kann. Der Einsatzstatus *NICHTS* steht für den Zustand vor dem Anschließen des Lungenautomaten. In dieser Zeit muss die voraussichtliche Einsatzzeit noch nicht feststehen und die Einsatzzeit nicht überwacht werden. Sobald der Lungenautomat angeschlossen wird, startet der Einsatz und der Einsatzstatus ändert sich zu *START*. Spätestens zum Zeitpunkt des Anschließens muss definiert werden, wie lange der Einsatz voraussichtlich dauert, denn die FwDV 7 schreibt vor, dass zu 1/3 und 2/3 der erwartenden Einsatzzeit eine Druckabfrage erfolgen muss. Der Countdown stellt eine Uhr dar, die diese Zeitpunkte bemisst, dieser darf zum Zeitpunkt des Einsatzstarts nicht null sein. Ein Trupp besteht aus mindestens

zwei Mitgliedern und maximal drei. Ein Trupp umfasst genau einen Truppführer und einen Truppmann, falls ein drittes Mitglied zum Trupp zugeordnet wird, hat dieser die Funktion des zweiten Truppmannes. Der Einsatzauftrag muss zum Zeitpunkt des Anschließens des Lungenautomaten feststehen. Daher muss auch zwingend der eigentliche Auftrag definiert sein. Eine Einsatzkraft hat einen Namen, der erfasst werden muss (siehe 2.1.1). Die zwingende Angabe der Funktion ergibt sich aus dem Umstand, dass der Trupp nur einen Truppführer haben darf und zwei Truppmitglieder. Ein PA-Gerät als Klassenattribut muss zudem maximal zu einem Exemplar der Klasse Einsatzkraft zugewiesen werden, da jede Einsatzkraft ihr eigenes PA-Gerät mit eigenen Parametern (Luftvorrat zu bestimmten Einsatzzeitpunkten) trägt. Der Pressluftatmer muss einen positiven maximalen Luftdruck haben und beim Initialisieren definiert sein. Ein PA-Gerät hat zu verschiedenen Zeitpunkten unterschiedliche Luftvorräte. Daher wird über die Methode setzeDruck der Druck für den entsprechenden Zeitpunkt definiert. Dieser muss zum Startzeitpunkt positiv und mindestens 90% des maximalen Nennfülldrucks erfüllen [7, S.18]. In allen anderen Fällen muss der Druck positiv sein. Ein Luftdruck (Rückzug-Soll) der Klasse PA-Gerät wird abgeleitet (derived) aus dem entsprechenden Einsatzgrundsatz der Rückwegsdruckberechnung nach 2.1.2.

3.3 Prozessmodellierung der Atemschutzüberwachung

Im Folgenden wird der Prozess der Atemschutzüberwachung in BPMN modelliert. Der Ablauf der Atemschutzüberwachung gestaltet sich wie folgt: die überwachende Person erfasst grundlegende Informationen zum Einsatzgeschehen, mindestens jedoch die Daten die nach 2.1.1 benötigt werden. Jeder Trupp, der aktiv in Einsatz geht, registriert sich von sich aus bei der Atemschutzüberwachung (Start-Event). Dabei wird das Einsatzziel, die Luftdrücke der Atemschutzflasche und die voraussichtliche Einsatzzeit festgehalten. Verschiedene eventbasierte Ereignisse, wie ein Funkspruch des Trupps oder das Erreichen von Anteilen der voraussichtlichen Einsatzzeit, erfordern das Eintragen von Luftdrücken der PA-Geräte in die Atemschutztafel. Die überwachende Person kalkuliert an der bekannten Formel 2.1.2, ob ein Rückzug nötig ist und teilt dies ggfs. dem Trupp mit. Die Überwachung endet mit der Mitteilung des Trupps, dass die Geräte abgelegt worden sind (End-Event). Notrufmeldungen wurden in dieser Modellierung nicht explizit hervorgehoben. Prinzipiell muss sich auch ein Sicherheitstrupp (besonderer Trupp, welcher zur Rettung verunfallter Kameraden ausgerüstet ist) bei der Atemschutzüberwachung

anmelden. Informationen für einen Notfall, wie die Verortung des verunfallten Trupps, sind bereits erfasst.

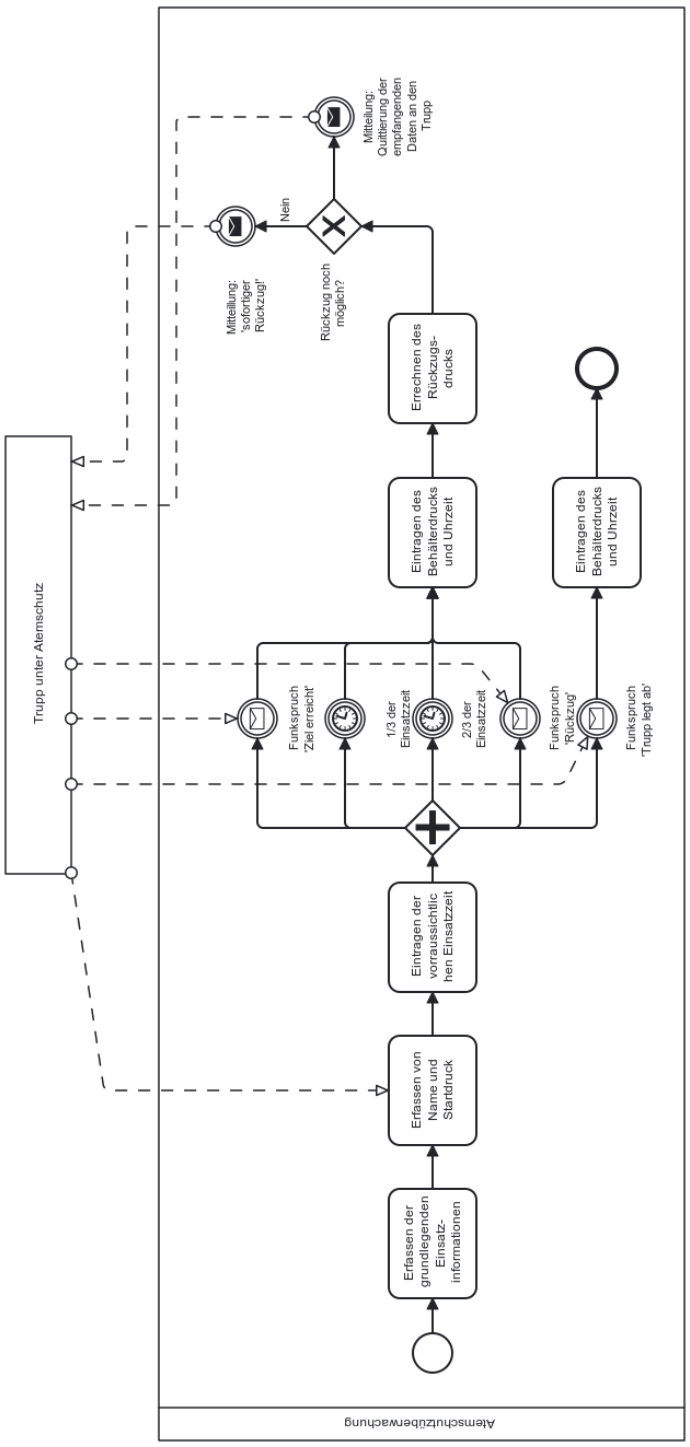


Abbildung 3.5: BPMN-Modellierung eines Einsatzablaufs

3.4 Auswahl der Technologie

Die Entwicklung der Applikation ist in der IDE Android Studio mit der Programmiersprache Java erfolgt. Insbesondere im mobilen Endgerätesegment der Tablets wurden vorzugsweise Tablets mit dem Betriebssystem Android gekauft (siehe 3.6) [39]. Alleine im Jahr 2020 wurden 108,1 Millionen Tablets mit dem Betriebssystem Android verkauft. Den zweitgrößten Absatz haben Tablets mit dem Betriebssystem iPadOS, hier wurden im Jahr 2020 57,6 Millionen Einheiten verkauft. Das stellt eine Differenz von 50,5 Millionen Einheiten dar. Folglich kann durch die Auswahl des Betriebssystems Android die größte Nutzerschaft im Tabletsegment erreicht werden. Für die Entwicklung von Android-Applikationen empfiehlt der Hersteller Google die IDE Android Studio [32, S.2]. Dabei lässt sich unter Android Studio mit der Programmiersprache Kotlin oder Java programmieren. Im Hinblick auf eine potentielle Weiterentwicklung empfiehlt sich eine Programmiersprache mit einer großen Entwicklerschaft. Java ist die am zweitmeisten genutzte Programmiersprache weltweit (Abbildung 3.7) [14]. Durch die große Entwicklerschaft bieten sich mehr Ressourcen, Unterstützung und eine umfangreichere Dokumentation.

Zum Testen der Anwendung wurde ein Samsung Galaxy Tab 4 (SM-T530) mit dem Betriebssytem Android 5.0.2 (Lolipop) verwendet.

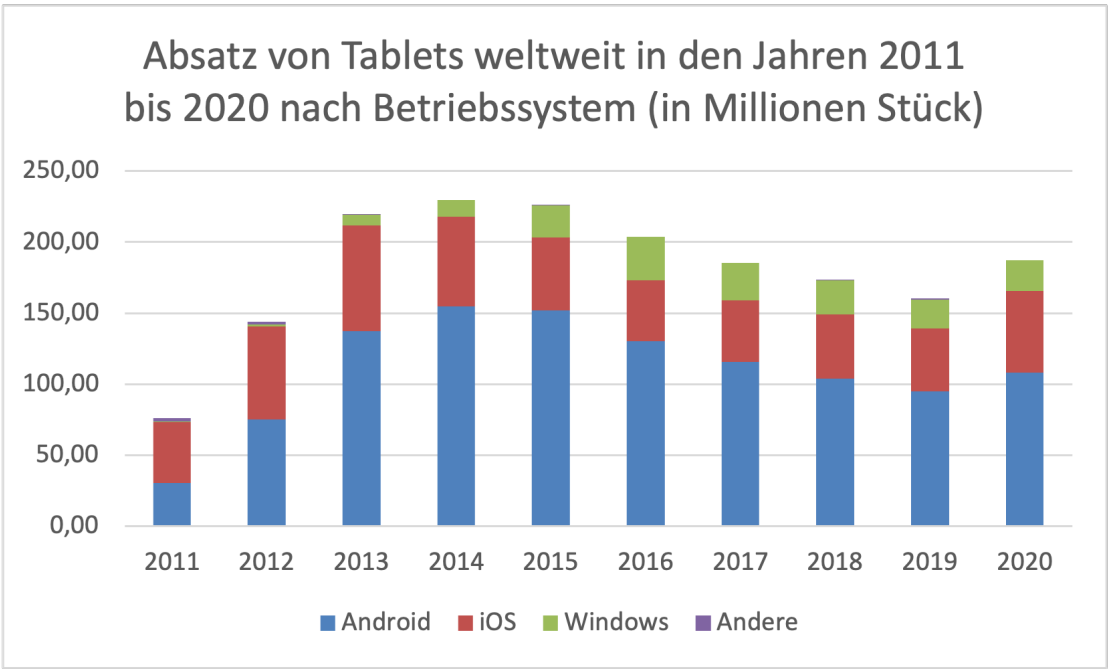


Abbildung 3.6: Absatz von Tablets weltweit in den Jahren 2011 bis 2020 nach Betriebssystem (in Millionen Stück)

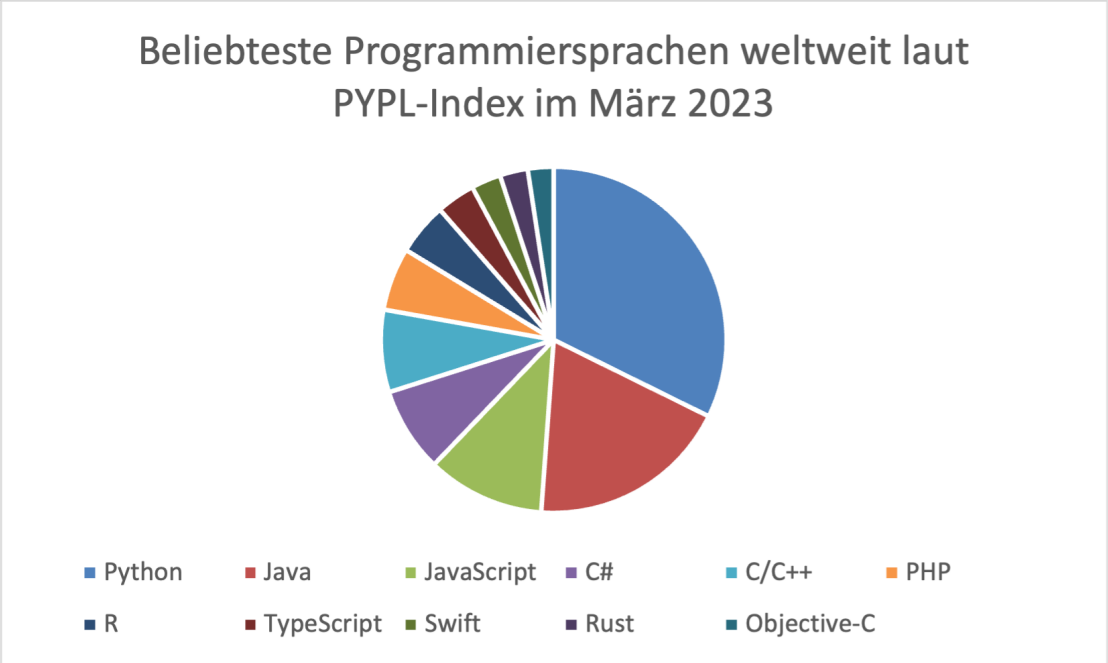


Abbildung 3.7: Beliebteste Programmiersprachen weltweit laut PYPL-Index im März 2023

4 Implementierung

Nachdem der Entwurf der Software abgeschlossen ist, geht es um die Umsetzung der Applikation. In diesem Kapitel wird behandelt, wie der Entwurf realisiert worden ist. Bei der Entwicklung mussten Dinge ergänzt werden, die nicht Teil des Softwareentwurfs sind, sowie Umstände, die bei der iterativen Softwarentwicklung zu Problemen geführt haben und wie die Probleme durch entsprechende Gegenmaßnahmen behoben werden konnten.

4.1 Architektur der Applikation

Betrachtet man das Domänenmodell und die Projektstruktur 4.1, dann fällt auf, dass die Paketnamen, sowie die Klassenstrukturen dem Domänenmodell entsprechen. Die Umsetzung der Klassen wurden so nah wie möglich am Entwurf getätigt. Dabei sind die mit Sternchen markierten Klassen, Klassen die im Entwurf noch nicht Bestandteil waren. Dabei handelt es sich um die Klassen, die die Schnittstelle zum Layout bilden (abgebildet im Paket Layout), Klassen die die Datenhaltung und somit die Schnittstelle zur Datenbank bilden (abgebildet im Paket Datenhaltung), sowie das Design-Pattern Dependency Injection[1] abgebildet durch den *InjectorManager* und eine Klasse zum Konvertieren von Einheiten *Converter*.

[1] Hierzu mehr im Unterabschnitt unter 4.2.2

```
.
|____InjectorManager.java*
|____Converter.java*
|____Layout*
| |____UeberwachungstafelAdapter.java*
| |____MainActivity.java*
| |____Ueberwachungstafel.java*
|____Kollektivdaten
| |____Trupp.java
| |____Einsatzauftrag.java
|____Kerndaten
| |____Einsatzinformationen.java
|____Datenhaltung*
| |____DatenverwaltungImpl.java*
| |____CronjobSpeichern.java*
| |____Datenverwaltung.java*
|____Personendaten
| |____Status.java
| |____Funktion.java
| |____Einsatzkraft.java
| |____PAGeraet.java
```

Abbildung 4.1: Projektstruktur der Applikation

4.2 Umsetzung der Anforderungen

Für die grafische Darstellung der Applikation wird eine XML-Datei mit Layoutelementen vorgegeben. Diese wird mithilfe der Superklasse *AppCompatActivity* der AndroidSDK an den Code gebunden. Zwei Klassen, die diese Klassen extenden, sind die Klasse *MainActivity* und *Ueberwachungstafel*. Dabei besitzt die *MainActivity* jedoch kein Layout und ist nur zur Initialisierung und Instantiierung der Klassen zuständig. Der Nutzer interagiert mit der Klasse *Ueberwachungstafel*. Diese stellt die einzige Nutzerschnittstelle zum System dar. Entsprechend dem Entwurf unter 3.2 wurden die einzelnen Klassen im System entwickelt. Dies lässt sich an der Paketstruktur unter 4.1 erkennen. Diese ist eine Abbildung der entworfenen Pakete unter 3.1. Für die Nutzerschnittstellen innerhalb der *Ueberwachungstafel* wurden entsprechende Dialoge erstellt, unter denen der Nutzer durch Knopfdruck auf das jeweilige Feld 4.5 Eingaben tätigen kann (Anforderung B.1.1). Soweit wie möglich wurde der Nutzer hier in seinen Eingaben limitiert. So ist die Angabe des Luftdrucks nur durch eine numerische Eingabe möglich und die alphanumerische Eingabe wird blockiert (Anforderung B.2.3). Für das Erfassen von bis zu drei Trupps werden intern drei Instanzen der Klasse *Trupp* vorgehalten (Anforderung B.1.2). Für das

Erfassen der Drücke in den jeweiligen Status wird intern gespeichert, wann das Feld das erste Mal erfasst worden ist und die aktuelle Systemzeit gespeichert (Anforderung B.1.3). Nach Ausfüllen der Status *START* und *ZIEL* wird der voraussichtliche Rückzugsdruck von der Klasse *PAGerät* errechnet (Anforderung B.1.4). Die Anforderungen B.1.8 bis B.1.13 beziehen sich auf die Zeiterfassung der einzelnen Trupps während des Einsatzes. Hierfür wurden drei *CountDownTimer* innerhalb der Klasse *Trupp* angelegt. Nachdem diese Timer abgelaufen sind, wird ein Dialog aufgerufen, der dem Nutzer mitteilt, dass eine Druckabfrage erforderlich ist. Visuell werden die Timer in der GUI umfangreich in der jeweiligen Truppansicht angezeigt. In klein wird die Zeit neben dem Truppnamen unten in der Truppleiste angezeigt. Der Nutzer wird nicht überhörbar akustisch und visuell benachrichtigt, wenn die maximale Einsatzzeit überschritten wird.

4.2.1 Bedienung der Applikation

Beim Aufrufen der Applikation gelangt der Anwender direkt zur Übersicht, von der aus die Überwachung gestartet werden kann (siehe Abbildung 4.5). In der unteren Zeile kann dieser zwischen drei zu überwachenden Trupps wechseln. Zusätzlich zum Namen des Trupps wird auch noch die Resteinsatzzeit des Trupps unten in der Anzeige angezeigt. In der obersten Zeile können grundlegende Informationen, die unter die Klasse *Einsatzinformationen* fallen, erfasst werden. Hierzu zählen u.a. die Einsatznummer sowie die Adresse. Über den Knopf *NEU* den Einsatz zurücksetzen. Damit dies nicht ausversehen passiert, wird der Nutzer nochmals um Bestätigung gebeten (siehe Abbildung 4.8), entsprechend der Anforderung an B.2.2. Der Nutzer kann durch Anklicken der einzelnen Felder in der Übersicht 4.5 die entsprechenden Felder ausfüllen. Mittels des Drucks auf den Knopf *START* kann der Nutzer unter Eingabe der voraussichtlichen Einsatzzeit die Timer starten, welche zu den gegebenden Zeitpunkten, welche unter 2.1.1 festgelegt sind, einen Hinweis zur Druckabfrage an den Überwachenden übermitteln (siehe Abbildung 4.6). Falls die voraussichtliche Gesamteinsatzzeit überschritten wird und der Einsatz nicht beendet worden ist, wird der Anwender von einem lauten Warnton und einem Dialog darauf hingewiesen. Falls die entsprechenden Luftvorräte am Startzeitpunkt und zum Erreichen des Ziels ausgefüllt worden sind, wird der Soll-Rückzugsdruck errechnet und hinter dem Truppmitglied angezeigt. Maßgebend sind nach 2.1.2 die höchsten errechneten Soll-Rückzugsdrücke. Diese werden gelb hinterlegt, sodass man mit einem Blick sehen kann, welcher Luftvorrat entscheidend für den gesamten Trupp ist (siehe Abbildung 4.9).

4.2.2 Verwendete Design-Pattern

Alle verwendenten Design-Pattern können dem Anhang unter C entnommen werden. Für die Implementierung wurde das Design-Pattern Dependency Injection (DI) eingesetzt. Der Einsatz dieses Pattern förderte zum einen die Kopplung der einzelnen Klassen zu reduzieren (da diese nicht mehr als Klassenvariablen übergeben werden müssen), zum anderen ermöglichte es eine einfache Implementierung von Interfaces, wie dem der *Datenhaltung*. Die Abstraktion zur konkreten Implementation der Datenbank in der Datenbankklasse *DatenhaltungImpl* ermöglicht hier die Möglichkeit die Datenbankimplementation, für mögliche zukünftige Versionen, schnell auszutauschen. Des weiteren wurde das Adapter-Pattern eingesetzt. Da dieses Pattern beim Testen eingesetzt worden ist, wird auf dieses Pattern erst im Abschnitt 5 eingegangen.

4.2.3 Anti-Pattern: Gottklasse

Während der iterativen Entwicklung des Codes ist die Layoutklasse *Ueberwachungstafel* zu einer Gottklasse (WMC[2] $>$ 47, ATFD[3] $>$ 5, TCC[4] $>$ 0,33) geworden. Eine Gottklasse entspricht einem Anti-Pattern, welches unter anderem die Code-Verständlichkeit reduziert und das Testen erschwert [40, S.50, S.55]. Um dem entgegenzuwirken wurden die Methoden fachlich in die jeweilig zugehörigen Klassen als Subroutinen ausgelagert. Zusätzlich wurde die McCabe-Metrik der zyklomatischen Komplexität eingesetzt (Formel 4.1), um zu große Methoden in entsprechende Subroutinen umzuwandeln und so das Codeverständnis zu verbessern [34].

$$M = E - N + 2P \quad \text{wobei} \begin{cases} E = \text{Anzahl der Kanten im Kontrollflussgraphen} \\ N = \text{Anzahl der Knoten im Kontrollflussgraphen} \\ P = \text{Anzahl der verb. Komponenten im Kontrollflussgraphen} \end{cases} \tag{4.1}$$

Die Verwendung der Metrik ist allerdings mehr als Richtwert zu verstehen, da bereits überschaubare Switch-Statements schnell eine hohe Code-Komplexität erreichen [29, S.100]. Im Kontext von Android kann bereits ein einfacher onClickListener der einen Dialog aufruft zu einer Komplexität von 6 führen (siehe Abbildung 4.2). Ein Beispiel, bei dem die

[2] Weighted Method Count (WMC)
[3] Access To Foreign Data (ATFD)
[4] Tight Class Cohesion (TCC)

```
1  button.setOnClickListener(v -> {
2  AlertDialog.Builder builder = new AlertDialog.Builder(this);
3  builder.setTitle("Titel");
4  builder.setMessage("Nachricht");
5  EditText input = new EditText(this);
6  builder.setView(input);
7  builder.setPositiveButton("Ok", new DialogInterface.OnClickListener()
       {
8      @Override
9      public void onClick(DialogInterface dialog, int which) {
10     //...
11     }
12 });
13 builder.setNegativeButton("Abbrechen", new DialogInterface.
       OnClickListener() {
14     @Override
15     public void onClick(DialogInterface dialog, int which) {
16         //...
17     }
18     });
19     AlertDialog dialog = builder.create();
20     dialog.show();
21 });
```

Abbildung 4.2: Beispielhafter onClickListener, mit AlertDialog und Lambda-Ausdruck, zyklomatische Komplexität von 6.

zyklomatische Komplexität, sowie die ATFD als Indikator für eine zu umfangreiche Methode benutzt wurde, ist der Code-Ausschnitt unter Abbildung 4.3. Hierbei ist besonders hervorzuheben, dass über die Klassen auf weitere Exemplare anderer Klassen zugegriffen wird. Zudem errechnet die Layout-Klasse hier für die Klasse PAGeraet, die fachlich selber dafür zuständig sein sollte, den Rückzugsoll-Druck. Die komplexe Methode wurde so geteilt, dass der Aufruf am PAGerat nicht mehr direkt erfolgt (Abbildung 4.4).

```
1  /** Ausschnitt des ersten Methodenentwurfs für einen onClickListner()
       innerhalb eines Lambda-Blocks der Layout-Klasse(!), um den Druck
      zu verändern **/
2  PAGeraet pag = aktuellerTrupp.get_trupp()[memberIndex].getPag();
3  pag.aktualisiereDruck(ZIEL, Integer.parseInt(text));
4  setzeUhrzeit(ZIEL);
5  if (pag.getDruck(START) != 0) {
6      int druckstart = pag.getDruck(START);
7      int druckziel = pag.getDruck(ZIEL);
8      int rueckzugsdruck = (druckstart - druckziel) * 2;
9      pag.aktualisiereDruck(Status.RUECKZUGSOLL, rueckzugsdruck);
10     rueckzugsollTextView.setText(pag.getDruck(Status.RUECKZUGSOLL) +
           " bar");
11     findeHöchstenDruck();
12     }
```

Abbildung 4.3: Erster Entwurf eines onClickListener im Rahmen eines MVPs. Der Code weist eine zyklomatische Komplexität von 6 und ein ATFD von 5 (pag hier als seperate Zugriffsmöglichkeit) auf.

```
1  //Code der Klasse Ueberwachungstafel
2
3   /**
4   * Diese Methode bestimmt den höchsten Druck im Rückzugsoll
5   * und markiert dieses Feld entsprechend.
6   * 0-Drücke werden nicht markiert.
7   */
8  protected void findeHöchstenDruck() {
9      List<Funktion> niedrigsteRueckzugswerte = aktuellerTrupp.
           gibNiedrigsteRueckzugswerte();
10     for (Funktion funktion : Funktion.values()) {
11         TextView textView = truppStatusFelder.get(funktion).get(
               Status.RUECKZUGSOLL);
12         textView.setBackgroundColor(Color.TRANSPARENT);
13     }
14     for (Funktion funktion : niedrigsteRueckzugswerte) {
15         TextView textView = truppStatusFelder.get(funktion).get(
               Status.RUECKZUGSOLL);
16         textView.setBackgroundColor(Color.YELLOW);
17     }
18     pruefeAufZuWenigRueckzugsdruck();
19 }
20 //Code der Klasse PAGeraet
21 /**
22  * Diese Methode errechnet den Soll-Rueckzugsdruck,
23  * wenn Start und Ziel bekannt sind.
24  */
25    private void errechneRueckzugsdruck() {
26     if (druecke.containsKey(Status.START) && druecke.containsKey(
           Status.ZIEL) && getDruck(START) != 0) {
27         int druckstart = getDruck(START);
28         int druckziel = getDruck(ZIEL);
29         int rueckzugsdruck = (druckstart - druckziel) * 2;
30         druecke.put(Status.RUECKZUGSOLL, rueckzugsdruck);
31     }
32 }
```

Abbildung 4.4: Korrigierter Code mit einer zyklomatischen Komplexität von 3 und einer ATFD von 2 (protected void findeHöchstenDruck()). Die Methodenaufrufe werden fachlich korrekt an der Klasse Trupp ausgeführt, die entsprechenden Methodenaufrufe an die Truppmitglieder in Subroutinen der Klasse PAGeraet weiter delegiert und gesammelt zurückgegeben.

Abbildung 4.5: Grundstruktur GUI

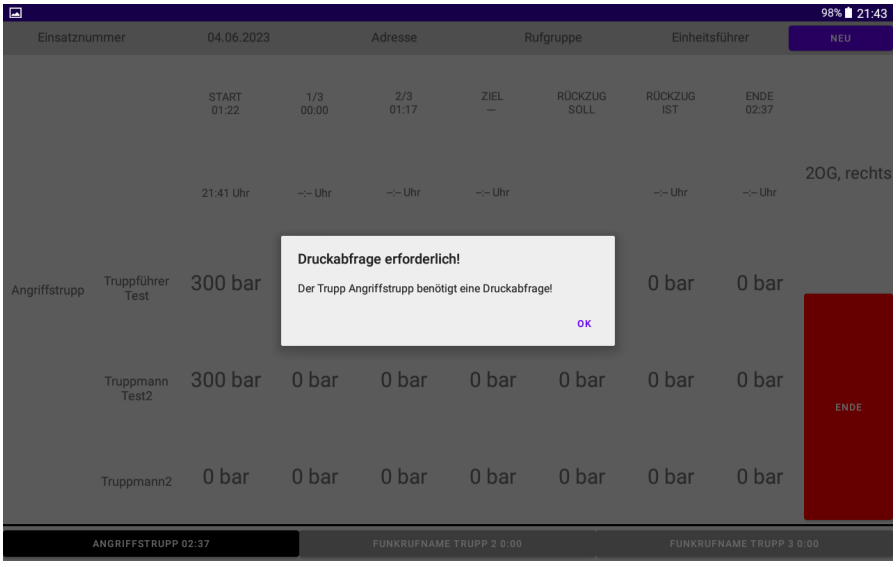


Abbildung 4.6: Druckabfrage im Einsatz

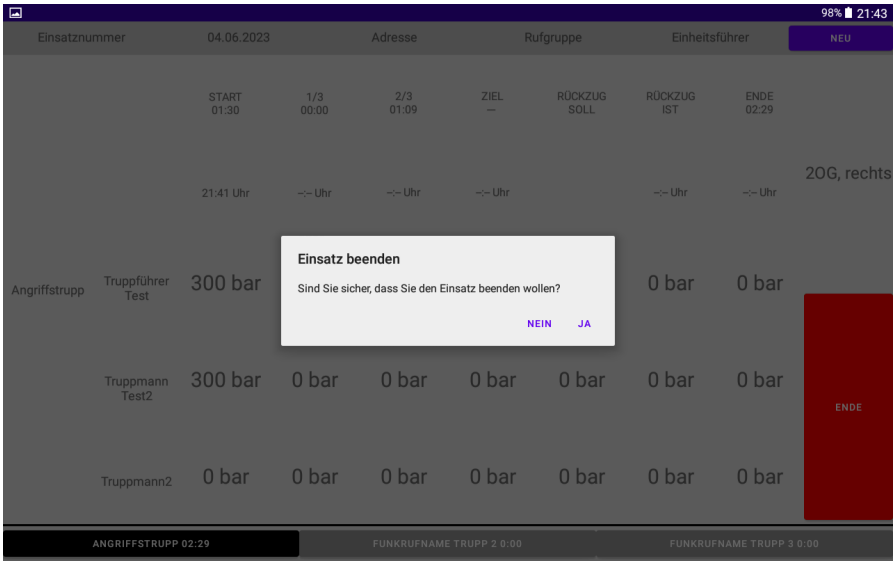


Abbildung 4.7: Sicherheitsmechanismus beim Druck von „ENDE“

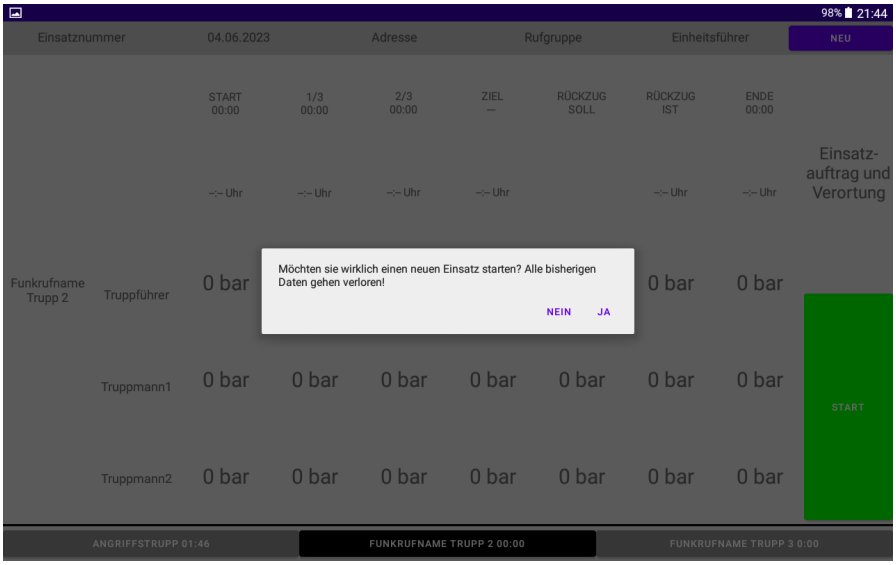


Abbildung 4.8: Sicherheitsmechanismus beim Druck von „NEU“

Abbildung 4.9: Die Applikation markiert automatisch den Richtwert für den Rückzugsdruck, der für den gesamten Trupp gilt.

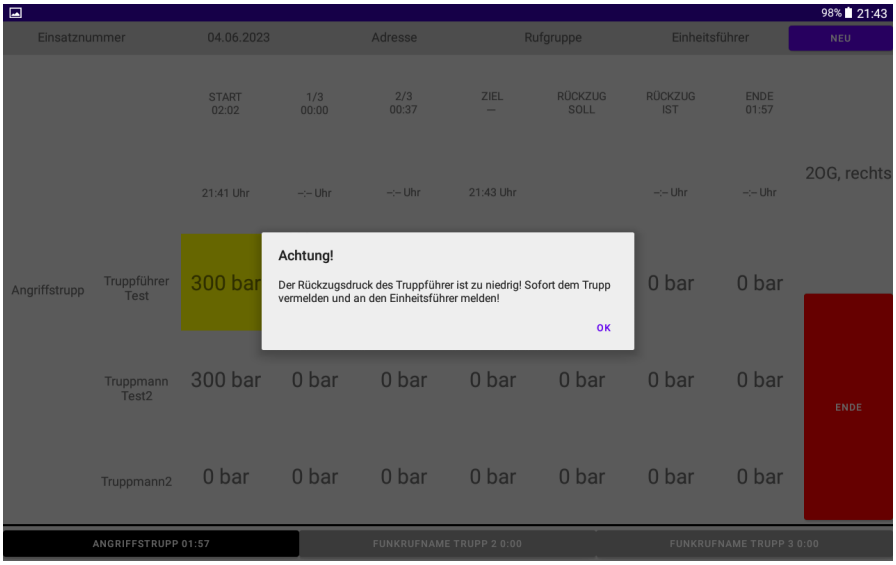


Abbildung 4.10: Falls ein bestimmter Schwellenwert unterschritten wird, bei dem kein Rückzug mehr möglich ist, wird die überwachende Person darauf hingewiesen und die entsprechenden Drücke markiert.

4.3 Integration und Schnittstellen

Durch die Entwicklung als Android-Applikation wird zum einen die Schnittstelle der AndroidSDK verwendet. Innerhalb der Applikation wird dabei auf Androidressourcen über die *R-Klasse*[5], als auch auf eine Datenbankschnittstelle *SQLiteOpenHelper*, die eine Verbindung zur Datenbank *SQLite* herstellt, zugegriffen. Im Folgenden wird erläutert, wie diese Ressourcen im Rahmen dieser Applikation genutzt worden sind.

4.3.1 Datenhaltung mittels SQLite

Die Anforderungen des Speicherns (B.1.5), Ladens (B.1.6) und zyklischen Speicherns (B.1.7) und daraus resultierend einer möglichen Wiederherstellung im Fehlerfall durch das Laden aus der Datenbank (B.2.4) wurden die Klassen des Pakets *Datenhaltung* entwickelt. Für die Datenhaltung wurde zunächst das Interface *Datenhaltung* entwickelt, welches in den InjectorManager (DI) eingebunden wurde. Dieser modulare Aufbau ermöglicht es, die Implementierung schnell auszutauschen, falls auf eine andere Schnittstelle, in zukünftigen Versionen, wie z.B. Room zurückgegriffen werden soll. Die konkrete Implementierung des Interfaces wurde in der Klasse *DatenverwaltungImpl* realisiert. Bei der Umsetzung wurde die SQLite-Instanz unter Android genutzt. Damit ACID-Eigenschaften [28, S.305] sichergestellt werden, wird zu Beginn des Lade- sowie Speichervorgangs eine Transaktion gestartet. SQLite erfüllt die ACID-Eigenschaften [22]. Zeitgesteuert wird alle fünf Minuten über die Klasse *CronjobSpeichern* das Speichern des aktuellen Datenbestands gestartet. Zusätzlich kann der Nutzer durch das Betätigen der Zurücktaste auch manuell das Speichern auslösen. Die Datenbank ist nach dem Modell unter 4.11 aufgebaut. Um eine SQL-Injection zu verhindern, wird eine Eingabe-Filterung vorgenommen [28, S.388-389].

4.3.2 Verwendung von Layoutressourcen

Die Schnittstelle zwischen grafischer Benutzeroberfläche und Backend wird mit verschiedenen Schnittstellen der AndroidSDK umgesetzt. Dabei greift ausschließlich die Klasse *Ueberwachungstafel* und *MainActivity* im Package *Layout* auf die Layoutschnittstellen zu. Dabei wurde ein XML-Layout an eine Klasse gebunden, die die Superklasse *AppCompatActivity* extended. Mittels der *R-Klasse* konnte auf die einzelnen Ressourcen, wie

[5] Die R-Klasse verlinkt unter Android grafische Ressourcen mit dem Javacode

TextView — die einzelne Textbausteine, oder *Button* die einzelne Buttons darstellen, zugegriffen werden. Die einzelnen Layoutelemente boten dabei schon Methoden an, um entsprechend Farben, Texte, Interaktionen mit diesen im Backend zu registrieren oder zu verändern.

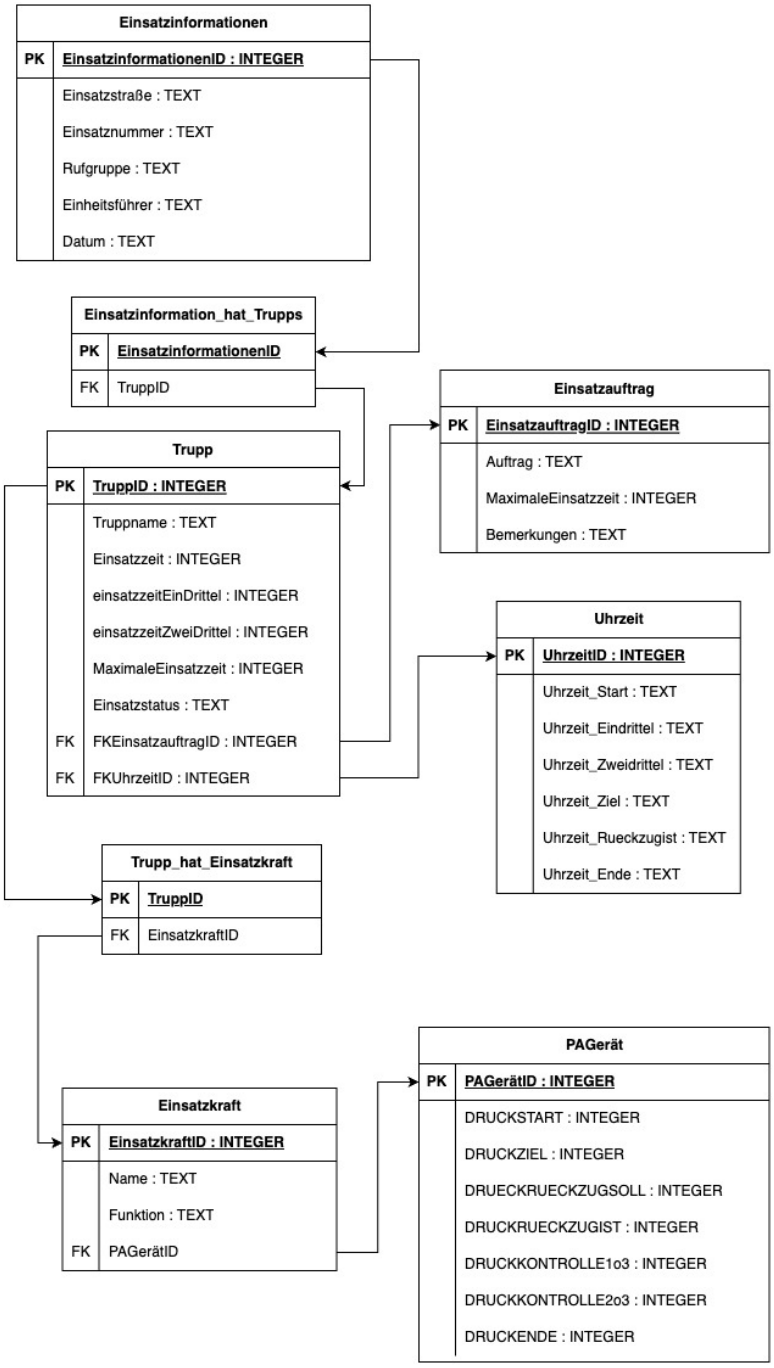


Abbildung 4.11: ER-Modell der Datenbank mit den Datentypen von SQLite

5 Testverfahren und Beurteilung

Das Testen war fester Bestandteil der iterativen Softwareentwicklung dieser Applikation. Es wurde dabei das Versionierungstool *Git* eingesetzt. Mithilfe des Aufsetzens einer CI-Pipeline, die die Tests beim Hochladen einer neuen Version im Git-Workflow automatisiert ausführen, wurden Fehler schnell entdeckt. Instrumented Tests, wie UI-Tests, die einen Emulator oder eine echte Systemumgebung zum Ausführen benötigen, wurden vor dem Pushen manuell ausgeführt. Dieser Bestandteil des Testprozesses ist auch unter 5.1 nochmal visualisiert dargestellt. Nach Möglichkeit wurden Unit-Tests nur unter Verwendung von JUnit geschrieben. Klassen, die im UML-Diagramm unter 3.2 entworfen worden sind, sind vollständig nur mit JUnit getestet worden. Als einzige Klasse wurde dafür bei der Klasse *Trupp* das Adapter-Pattern eingesetzt. Mithilfe des Adapter-Pattern konnte die Klasse von der Verbindung mit der Android Framework über die Klasse *Ueberwachungstafel* gelöst werden. Klassen, die das Android-Framework zum Testen benötigen, wie *Ueberwachungstafel* oder *DatenverwaltungImpl*, wurden mithilfe von Roboeletric getestet. Beim Testen wurde nach dem Schema der Android-Testpyramide 5.2 vorgegangen [26]. Die Testpyramide ist ein Konzept, das besagt, dass die Mehrheit der Tests in einer Softwareentwicklung aus Unit-Tests bestehen sollte, gefolgt von Integrationstests und einem kleineren Anteil an Systemtests. Dabei ist der Begriff Integrationstest in diesem Konzept nicht genau gefasst, weshalb eine projektabhängige Definition des Begriffes wichtig ist. Ein Integrationstest stellt in diesem Projekt einen Test dar, der zwischen mindestens zwei oder mehr Klassen agiert. So ist zum Beispiel der Test von *DatenverwaltungImpl*, welcher zwingend eine Instanz der *Einsatzinformationen* benötigt, um die Daten zu speichern, ein solcher Test. Für die Erstellung der Unit- und Integrationstests sind die Test-Pattern Äquivalenzklassentest, als auch Arrange-Act-Assert eingesetzt [27][10, S.97]. Beispielhaft können Tests, die diese Pattern erfüllen dem Anhang unter D entnommen werden. Bei den getätigten Systemtests wird in der Arbeit zwischen manuellen Systemtests und automatisierten Systemtests unterschieden. So wurde für die automatisierten Systemstests das Framework Espresso oder Application Exerciser Monkey eingesetzt.

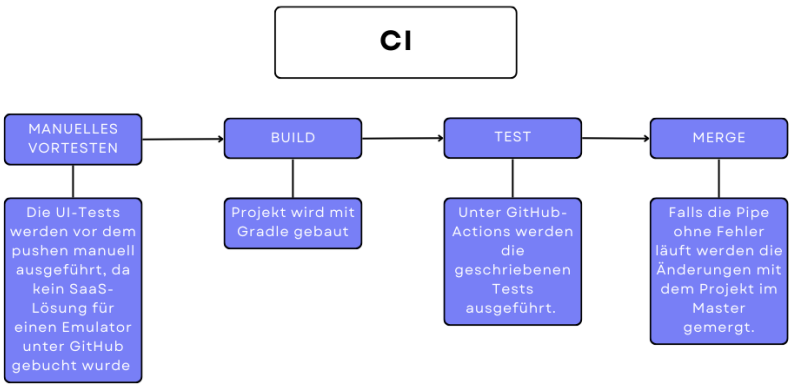


Abbildung 5.1: Testprozess während der Entwicklung

Ein manueller Systemtest ist die Validierung des unter 3.5 erzeugten BPMN-Modells im unteren Abschnitt.

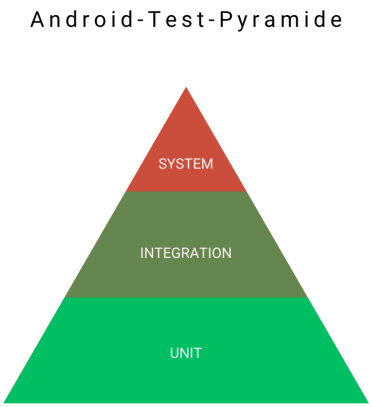


Abbildung 5.2: Android-Test-Pyramide nach [26]

5.1 Verwendung von JaCoCo

Als Richtwert für die Codeabdeckung mittels der bereits umgesetzten Testfälle wurde die Metrik Code Coverage zur Hilfe genommen. Zur Ermittlung der Metrik wurde das Tool JaCoCo verwendet. Während die Verwendung des Coverage-Tools unter reinen JUnit-Tests funktionierte, ist der Einsatz mit dem Test-Framework Roboeletric nicht erfolgreich. Die Code-Coverage wurde bei Testklassen, die Robeletric als Test-Runner verwendet haben, nicht gemessen. Ein offenes Issue im Repository von Roboeletric aus dem Jahre 2017 zeigte ähnliche Probleme [5]. Tatsächlich führte das Ausführen der Tests ohne Code-Coverage zu keinen Problemen, weshalb die Klassen, die Roboeletric nutzten für die Code-Coverage-Metrik ausgeschlossen worden sind. Für alle anderen Testklassen wurde die Metrik verwendet, um zu evaluieren, wie häufig und in welchem Umfang die Methode bereits getestet wurde.

5.2 Manueller Systemtest

Es wurde ein Systemtest anhand des entworfenen BPMN-Prozesses durchgeführt. Dabei wurde evaluiert, ob die Prozessschritte, welche bei der Atemschutzüberwachung erfolgen müssen auch in der Applikation dargestellt werden. Hierfür wurde sich ein fiktives

Szenario mit zeitlichen Abläufen überlegt, welches dem BPMN-Prozessablauf entspricht. Der entwickelte Test kann dem Anhang unter D entnommen werden. Alle Schritte wurden innerhalb des Entwicklungsprozesses dargestellt. Dabei wurde der Schritt *Errechnen des Rückzugsdrucks* (2.1.2) sogar optimiert, da der Nutzer hier nicht mehr händisch den Druck errechnen muss, sondern die Applikation dies vollautomatisch übernimmt.

5.3 Ergebnisse und Diskussion

In der Testphase wurden diverse Aspekte der Software geprüft. Ein primärer Fokus lag auf der ordnungsgemäßen Funktion der Software, dies ist jedoch nur eine Facette des Evaluierungsprozesses. Zentral waren auch die Anforderungen an die Robustheit der Software, die in den Punkten B.2.2, B.2.3, und B.2.4 detailliert beschrieben und mit Espresso und Application Exerciser Monkey umgesetzt sind. Ziel war es, sicherzustellen, dass das System Fehleingaben effektiv vermeidet und dabei gleichzeitig stabil bleibt. Es ist allerdings wichtig zu betonen, dass die Usability der Software in diesem Durchgang nicht geprüft wurde. Obwohl die Benutzerfreundlichkeit unter den geforderten Anforderungen B.2.1 aufgeführt war, wurde diese spezielle Komponente nicht in die aktuelle Testrunde aufgenommen. Zukünftige Tests könnten sich darauf konzentrieren, diese zu gewährleisten. Mithilfe des manuellen Systemtests konnte evaluiert werden, dass die Software den Entwurfsbedingungen entspricht und sogar hinsichtlich der Errechnung des Rückzugsdrucks B.1.4 optimiert worden ist.

6 Zusammenfassung und Ausblick

Im abschließenden Kapitel dieser Arbeit liegt der Schwerpunkt auf der Evaluierung der zentralen Fragestellung, die zu Beginn gestellt wurde. Hierbei werden die gewonnenen Ergebnisse kritisch betrachtet und die Methoden und Herangehensweisen, die zur Beantwortung der Forschungsfrage angewandt wurden, reflektiert. Im Anschluss wird ein Ausblick auf die Zukunft gewagt, indem mögliche Schnittstellen und Erweiterungsmöglichkeiten der Arbeit dargelegt werden. Damit soll eine Vorstellung vermittelt werden, wie die vorliegende Arbeit fortgeführt und vertieft werden könnte, um ihren Nutzen und ihre Bedeutung in der weiterführenden Forschung zu steigern.

6.1 Zusammenfassung der Ergebnisse

Die zentrale Fragestellung zu Beginn dieser Arbeit war, in welchem Ausmaß eine digitale Applikation die manuelle Atemschutzüberwachung bei Feuerwehreinsätzen unterstützen kann und welche spezifischen Anforderungen dafür notwendig sind. Die Ergebnisse der Arbeit haben gezeigt, dass eine Digitalisierung der manuellen Atemschutzüberwachung durchaus umsetzbar ist. Besonders bemerkenswert ist dabei die Tatsache, dass durch den Systemtest, welcher die Software anhand des BPMN-Modell validiert (siehe Abschnitt 5.2), bestimmte Prozesse, wie die Berechnung des Rückzugdrucks (siehe Abschnitt 2.1.2) sogar optimiert. Die zur Umsetzung nötigen Anforderungen wurden streng nach der FwDV 7 mittels OCL-Constraints (siehe Anhang A) und einem speziellen Anforderungskatalog (siehe Abschnitte 3.1, B) definiert. Basierend auf diesen Anforderungen wurde die digitale Applikation entwickelt (siehe Abschnitt 4) und im Anschluss auf die Erfüllung der definierten Anforderungen getestet (siehe Abschnitt 5). Es wurden jedoch auch Bereiche identifiziert, in denen die Applikation in zukünftigen Arbeiten noch weiter verbessert werden kann. Im Abschnitt 6.3 sind potenzielle Ansatzpunkte für diese weiterführenden Arbeiten dargestellt.

6.2 Retrospektive

Im Rahmen einer kritischen Reflexion wird nun auf den Verlauf der Arbeit eingegangen und Ereignisse hervorgehoben, die im Prozess der Umsetzung dieser Arbeit hinderlich waren oder die Umsetzung unterstützt haben.

6.2.1 Auswahl der Programmiersprache

Unter 3.4 wurde erläutert, warum die Wahl der Programmiersprache auf Java fiel, wie erläutert, ist unter den Gesichtspunkten der Verbreitung der Sprache und der daraus resultierenden guten Dokumentation im Internet die Auswahl und Verwendung der Sprache eine gute Wahl. Als Alternative zu Java bietet die Android Studio IDE auch noch die Verwendung der Sprache Kotlin an. Insbesondere im Hinblick auf die Reduzierung von Boiler-Plate Code stellt Kotlin eine bessere Wahl dar als Java. Ein Beispiel, bei dem dies ersichtlich wird, ist im Anhang E hinterlegt. Seit 2017 empfiehlt Google die Nutzung der Programmiersprache Kotlin für die Android-App Entwicklung. 50% der in Android entwickelten Apps nutzen die Programmiersprache Kotlin [32, S.1]. Für folgende Arbeiten sollten Abwägungen geschaffen werden, welche Programmiersprache in welchem Umfang eingesetzt werden soll. Im AndroidSDK kann klassenweise gemischt in Kotlin und Java programmiert werden. Dies kann zum einen den angesprochenen Boiler-Plate Code reduzieren, zum anderen kann der Vorteil, der in diesem Abschnitt bereits oben genannt wurde, trotzdem noch genutzt werden.

6.2.2 Vorgehensmodell bei der Entwicklung der Applikation

Die Applikation wurde iterativ entwickelt. Dabei war das erste Ziel ein MVP[1] zu erzeugen. Durch das Nichteinhalten von Separation of Concerns (SoC) ist der Code sehr schnell sehr unübersichtlich geworden. Das Ergebnis dieser Vorgehensweise ist unter 4.2.3 bereits erläutert worden. In Hinblick auf zukünftige Entwicklungsprozesse kann dieses Vorgehen durch Aufnahme vom frühzeitigen Refactoring in den agilen Entwicklungsprozess stark minimiert werden. Auch die Verwendung von Single Responsibility Principle (SRP), bei der jede Softwareeinheit nur für eine präzise definierte Verantwortlichkeit zuständig sein sollte, kann hier helfen [24]. Divide et Impera, einem sehr berühmten Grundprinzip der Informatik, welches nicht nur bei Algorithmen zur Geltung kommt, sondern bei den

[1] MVP: Minimum Viable Product

Klassen, die Services oder Funktionalitäten auf einer hohen Abstraktionsebene anbieten, sollten aus Klassen bestehen, die Teilfunktionalitäten auf einer darunterliegenden Abstraktionsebene umsetzen. Durch kleinere Teilprobleme, die mit Klassen und Methoden gelöst werden erhöht sich das Codeverständnis stark, denn der Leser von Klassen und Methoden muss sich nunmehr auf kleinere Probleme konzentrieren. Die Einhaltung dieser Grundprinzipien kann dazu führen große Refactoringprozesse, in zukünftigen Projekten, zum Ende einer Entwicklungsphase zu vermeiden.

6.2.3 Entwurf von OCL-Constraints mittels USE

Die Verwendung von USE zeigte bereits in der frühen Entwicklungsphase die Bedeutung der Validierung von OCL-Constraints. Durch die interaktive Manipulation von Systemzuständen gab es ein direktes Feedback, ob der Systemzustand valide ist oder nicht. Beim Erstellen wurde bereits die Syntax geprüft, denn USE lässt fehlerhafte Syntax nicht zu. Getrennt konnte man die Assoziationen, als auch die Invarianten und Pre-/Postconditions prüfen. Fazit: USE beschleunigte den Entwurfsprozess der OCL-Constraints enorm. Im Rahmen der Entwicklung und der Testphase erwiesen sich die entwickelten OCL-Constraints als hilfreich. Sie erlaubten eine stärkere Ausrichtung auf die praktische Umsetzung, indem sie eine formale Definition verschiedener Bedingungen innerhalb der FwDV 7 lieferten. Aufgrund dieser Formalisierung wurden während des Entwicklungsprozesses weniger konzeptionelle Überlegungen hinsichtlich der Umsetzung der Bedingungen in FwDV 7 notwendig. In der Testphase konnten die OCL-Constraints effizient verwendet werden, um die Einhaltung der definierten Bedingungen zu überprüfen. Die OCL-Constraints lieferten durch ihre Einschränkungen ein festgelegtes Prüfschema, innerhalb dessen erwartet wird, dass bestimmte Bedingungen zu einem positiven oder negativen Ergebnis führen.

6.2.4 Verwendung von Design-Pattern

Durch die Verwendung des Android-Framework und den mehrfach nötigen Zugriff auf die Schnittstelle der AppCompatActivity stellte sich insbesondere die Dependency Injection als hilfreich heraus, da diese nicht über den Konstruktor als Klassenvariable an die entsprechenden Klassen übergeben werden musste, sondern der direkte Zugriff über die Klasse, die das Dependency-Injection-Pattern umgesetzt hat, ermöglicht wurde.

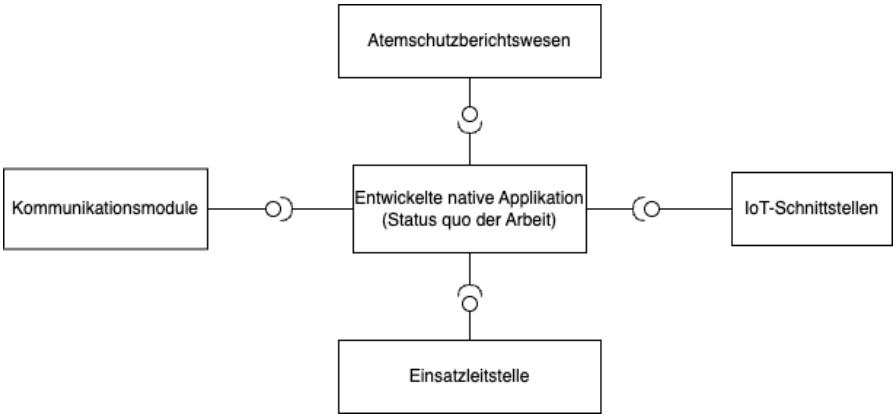


Abbildung 6.1: Mögliche Schnittstellen für eine Erweiterung der Applikation

6.3 Empfehlungen für künftige Arbeiten

Die in dieser Arbeit entwickelte Applikation erfordert eine Person als Atemschutzüberwachung, die die erforderlichen Daten in das System eingibt. Dabei beschränkt sich die Applikation auf die Erfassung von Daten im Einsatz zu Überwachungszwecken. In Hinblick auf eine weitere Digitalisierung sind folgende Schnittstellen (Abbildung 6.1) für das entwickelte System denkbar:

Der sogenannte Atemschutznachweis muss von jedem Atemschutzgeräteträger geführt werden [7, S.24]. Die FwDV sieht auch eine Umsetzung durch eine zentrale Lösung als eine Möglichkeit zur Führung dieses Nachweises vor. Eine Anbindung an eine Feuerwehrverwaltungssoftware über eine Schnittstelle, um die erforderlichen Daten zu erfassen und zu speichern, wäre eine mögliche Schnittstelle für das System, um den Prozess der Dokumentation zu vereinfachen.

Große Einsatzstellen haben oft einen hohen Funkverkehr [35, S.3]. Insbesondere beim Einsatz von Atemschutzgeräteträgern birgt dies ein großes Risiko, da es wichtig ist, dass jederzeit die Kommunikation klar und verständlich bleibt. Zudem ist die Kommunikation über eine Sprechmembran des Atemanschlusses schwierig [13, S.1]. Eine Rückmeldung des Trupps zur Lage (am Ziel angekommen, Rückzug angetreten) mittels Knopfdruck auf einem Drucktaster o.ä. kann die Funkkommunikation reduzieren. Ergänzend hierzu könnte ein IoT-Manometer die Drücke der Truppmitglieder übertragen.

Verschiedene IoT-Schnittstellen wurden bereits unter 2.2.5 vorgestellt. Auch diese Schnittstellen stellen eine mögliche Erweiterung der Applikation dar. Im Hinblick auf die Zuver-

lässigkeit, welche auch im o.g. Kapitel diskutiert wurde, sollten diese Schnittstellen aber nur unterstützend und nicht als vollständiger Ersatz eingesetzt werden.

Während beim Absetzen eines Notrufs im TMO-Modus der Funkgeräte ein Notruf automatisch auch an die Rettungsleitstelle (RLST)[2] weitergeleitet wird — ist dies nicht der Fall im DMO-Modus [38, S.20]. Damit eine vereinfachte Vorrückmeldung an die RLST zu einem Notruf in diesem Falle übermittelt werden kann, wäre es denkbar über ein GSM-Modul die Daten aus der Atemschutzüberwachung zu dem verunfallten Trupp zu übermitteln.

[2]Der Begriff Rettungsleitstelle wird hier synonym zu Einsatzleitstelle in Abbildung 6.1 verwendet.

Literaturverzeichnis

[1] Android Developers: *AndroidJUnitRunner*. – URL https://developer.android.com/training/testing/instrumented-tests/androidx-test-libraries/runner

[2] Android Developers: *Build local unit tests*. – URL https://developer.android.com/training/testing/local-tests

[3] Android Developers: *Espresso*. – URL https://developer.android.com/training/testing/espresso

[4] Android Developers: *UI/Application Exerciser Monkey*. – URL https://developer.android.com/studio/test/other-testing-tools/monkey

[5] Anonym: *Coverage fails with Robolectric tests in AndroidStudio*. – URL https://github.com/robolectric/robolectric/issues/3023

[6] Ausschuss Feuerwehrangelegenheiten, Katastrophenschutz und zivile Verteidigung (AFKzV): *Führung und Leitung im Einsatz – Führungssystem (FwDV 100)*. – URL https://hlfs.hessen.de/sites/hlfs.hessen.de/files/2022-09/FwDV%20100%20-%20Homepage.pdf

[7] Ausschuss Feuerwehrangelegenheiten, Katastrophenschutz und zivile Verteidigung (AFKzV): *Atemschutz (FwDV 7)*. Verlag W. Kohlhammer, 2002. – ISBN 9783555020594

[8] Ausschuss Feuerwehrangelegenheiten, Katastrophenschutz und zivile Verteidigung (AFKzV): *Einheiten im Lösch- und Hilfeleistungseinsatz (FwDV 3)*. Verlag W. Kohlhammer, 2008. – ISBN 9783555014357

[9] Baden-Württemberg Innenministerium: *Unfall-Bericht zum Einsatz Tübingen Reutlinger Straße 34/1*. Dec 2005. – URL https://www.atemschutzunfaelle.de/download/Unfaelle/u20051217-tuebingen-bericht-unfallkommission.pdf

[10] Beck, Kent: *Test Driven Development. By Example (Addison-Wesley Signature)*. Addison-Wesley Longman, Amsterdam, 2002. – ISBN 0321146530

[11] Beneke, Nils: *Das Feuerwehr-Lehrbuch Grundlagen - Technik - Einsatz*. Verlag W. Kohlhammer, 2017. – ISBN 9783170321717

[12] Bock ; Cook ; Rivett ; Rutt ; Seidewitz ; Selic ; Tolbert: *UML Unified Modeling Language Version 2.5.1*. Dec 2017. – URL https://www.omg.org/spec/UML/2.5.1/About-UML/

[13] Brodersen, Michael: *Signalverarbeitung für Kommunikationssysteme von Atemschutzvollmasken*, Christian-Albrechts-Universität zu Kiel, Dissertation, 2022. – URL https://macau.uni-kiel.de/receive/macau_mods_00003336

[14] Carbonnelle, Pierre: *Die beliebtesten Programmiersprachen weltweit laut PYPL-Index im April 2023*. Apr 2023. – URL https://de.statista.com/statistik/daten/studie/678732/umfrage/beliebteste-programmiersprachen-weltweit-laut-pypl-index

[15] CheckBox Soft- und Hardware KG: *rescueTABLET Datenblatt*. – URL https://rescuetablet.de/documents/rescueTABLET_Datenblatt.pdf

[16] Cimolino, Ulrich: *Atemschutz-Notfallmanagement Organisation, Ausbildung und Ausrüstung für Sicherheitstrupps und Schnelleinsatzteams*. ecomed Sicherheit, 2010. – ISBN 9783609774848

[17] Cimolino, Ulrich ; Aschenbrenner, Dirk ; Lembeck, Thomas ; Suedmersen: *Atemschutz sicheres und effizientes Vorgehen, Suchverfahren, Notfalltraining*. ecomed Sicherheit, 2004. – ISBN 9783609686639

[18] DIN Deutsches Institut für Normung e. V.: *DIN 14530-5: Löschfahrzeuge - Teil 5: Löschgruppenfahrzeug LF 10*. November 2019. – URL https://dx.doi.org/10.31030/3069606

[19] Dräger Safety AG: *Dräger PSS® Merlin® System*. – URL https://www.draeger.com/Products/Content/pss-merlin-pi-9041357-de.pdf

[20] Dräger Safety AG: *Dräger REGIS® 300 und 500 Atemschutzüberwachung*. – URL https://www.draeger.com/Products/Content/regis-300-und-500-pi-9041405-de-de.pdf

[21] Farhan, Laith ; Shukur, Sinan T. ; Alissa, Ali E. ; Alrweg, Mohmad ; Raza, Umar ; Kharel, Rupak: A survey on the challenges and opportunities of the Internet of Things (IoT). In: *2017 Eleventh International Conference on Sensing Technology (ICST)*, IEEE, 2017, S. 1–5. – URL https://ieeexplore.ieee.org/abstract/document/8304465

[22] Feiler, Jesse: *Defining SQLite*. S. 10. In: *Introducing SQLite for Mobile Developers*, APress, 2015. – ISBN 978-1-4842-1765-8

[23] Hamann, Lars: *USE Manual Version 0.1*. – URL https://github.com/useocl/use/blob/master/manual/main.md

[24] Haoyu, Wang ; Haili, Zhou: Basic Design Principles in Software Engineering. In: *2012 Fourth International Conference on Computational and Information Sciences*, 2012, S. 1251–1254

[25] Hilken, Frank ; Hamann, Lars: *History of the USE Tool 20 Years of UML/OCL Modeling Made in Germany*. 2020. – URL https://www.jot.fm/issues/issue_2020_03/article20.pdf

[26] Hirsch, Thomas ; Schindler, Christian ; Müller, Matthias ; Schranz, Thomas ; Slany, Wolfgang: An Approach to Test Classification in Big Android Applications. In: *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, IEEE, 2019, S. 300–308. – URL https://ieeexplore.ieee.org/document/8859509

[27] Jorgensen, Paul C.: *Equivalence Class testing*. S. 99–112. In: *Software Testing A Craftsman's Approach Fourth Edition*, CRC Press, 10 2013. – ISBN 9781466560697

[28] Kemper, Alfons ; Eickler, André: *Datenbanksysteme — Eine Einführung*. Bd. 10. De Gruyter Oldenbourg, 2015. – ISBN 9783110443752

[29] Koss, Daniel: Eine Komplexitätsmetrik basierend auf der kognitiven Wahrnehmung des Menschen. In: Unger, Herwig (Hrsg.): *Echtzeit 2020*. Wiesbaden : Springer Fachmedien Wiesbaden, 2021, S. 99–108. – ISBN 978-3-658-32818-4

[30] Lorenzen, Lars: *Unfallverhütung im Atemschutzeinsatz*. Kohlhammer Verlag, 2011. – ISBN 9783170213647

[31] Lüssenheide, Björn: *Feuerwehr-Dienstvorschrift 7 - Atemschutz*. Nov 2013. – URL https://www.atemschutzunfaelle.de/recht/fwdv7.html

[32] Mateus, Bruno G. ; Martinez, Matias: On the Adoption, Usage and Evolution of Kotlin Features in Android Development. In: *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. New York, NY, USA : Association for Computing Machinery, 2020 (ESEM '20), S. 1–2. – URL https://doi.org/10.1145/3382494.3410676. – ISBN 9781450375801

[33] Maurer ; Cimolino ; Rechenbach ; Kircher ; Schwägergen ; Kolberg ; Jungverdorben ; Backes: *Unfallkommission Einsatz Kierberger Straße 15*. Apr 1996. – URL https://www.atemschutzunfaelle.de/download/Unfaelle/Stampe/u19960306-koeln-abschlussbericht.pdf

[34] McCabe, T.J.: A Complexity Measure. In: *IEEE Transactions on Software Engineering* SE-2 (1976), Nr. 4, S. 308–320. – URL https://ieeexplore.ieee.org/document/1702388

[35] Ministerium des Inneren, für Digitalisierung und Kommunen Baden-Württemberg: *Eckpunkte zur Einführung des digitalen Einsatzstellenfunks bei den Feuerwehren in Baden-Württemberg*. Feb 2023. – URL https://www.lfs-bw.de/fileadmin/LFS-BW/themen/funk/digitalfunk/regelungenbetriebshandbuch/dokumente/Digitalfunk_Eckpunkte_Einsatzstellenfunk_Feuerwehr.pdf

[36] Object Management Group: *OMG Object Constraint Language Version 2.4*. Feb 2014. – URL https://www.omg.org/spec/OCL/2.4/About-OCL/

[37] Rossnagel, A. ; Jandt, S. ; Skistims, H. ; Zirfas, J.: *Zulässigkeit von Feuerwehr-Schutzanzügen mit Sensoren und Anforderungen an den Umgang mit personenbezogenen Daten*. – URL https://www.baua.de/DE/Angebote/Publikationen/Berichte/F2278.pdf?__blob=publicationFile&v=1

[38] Schleswig-Holstein, Landesfeuerwehrschule: *Sprechfunkausbildung Digitalfunk BOS*. Nov 2019. – URL https://www.digitalfunk-sh.de/DFSH/userfiles/files/Ausbildung/Teilnehmerunterlage_Sprechfunk_SH18.pdf

[39] Strategy Analytics, IDC;: *Absatz von Tablets weltweit in den Jahren 2011 bis 2020 nach Betriebssystem*. Jan 2021. – URL https://de.statista.com/statistik/daten/studie/220401/umfrage/prognose-zum-absatz-von-media-tablets-weltweit/

[40] Vasileva, Anna ; Schmedding, Doris: Vom Clean Model zum Clean Code. In: Oberweis, Andreas (Hrsg.) ; Reussner, Ralf (Hrsg.): *Modellierung 2016*. Bonn : Gesellschaft für Informatik e.V., 2016, S. 45–60. – URL https://dl.gi.de/bitstream/handle/20.500.12116/829/45.pdf

[41] Xtreme Labs, Pivotal Labs: *Roboeletric*. – URL https://robolectric.org/

A OCL-Constraints

Enumerations

```
model Atemschutzueberwachung

--enumerations
    enum Status {START, ZIEL, RUECKZUGSOLL, RUECKZUGIST, KONTROLLE1o3
        , KONTROLLE2o3, ENDE, NICHTS}
    enum Funktion {TRUPPFUEHRER, TRUPPMANN, TRUPPMANN2}
```

Klassen

```
--klassen
    class Datum
    -- Darstellung einer Klasse, welche ein Datum beinhaltet (dd.MM.
        yyyy)
    end

    class Countdown
    -- Darstellung einer Klasse, die ein Countdown hh:mm:ss darstellt
         und Start,Stop,End-Funktionalitäten besitzt.
    end

    class Einsatzinformationen
    attributes
        EinsatzstrasseundHausnummer : String
        Einsatznummer : Integer
        Rufgruppe : String
        Einheitsfuehrer : String
        Datum : Datum
    end


```

```
20      class Trupp
21      attributes
22          Funkrufname : String
23          Einsatzstatus : Status
24          Timer : Countdown
25      end
26
27      class Einsatzkraft
28      attributes
29          Name : String
30          Funktion : Funktion
31      end
32
33      class PAGeraet
34      attributes
35          maximalerDruck : Integer
36          rueckzugsolldruck : Real derived = ( self.wert[#START].druck
                - self.wert[#ZIEL].druck ) * 2
37      operations
38          setzeDruck(status : Status, wert : Integer)
39          getRueckzugSollDruck(): Real = ( self.wert[#START].druck -
                self.wert[#ZIEL].druck ) * 2
40      end
41
42      class Einsatzauftrag
43      attributes
44          Auftrag : String
45          maximaleEinsatzzeit : Integer
46          Bemerkungen : String
47      end
48
49      class Messwert
50      attributes
51          druck : Integer
52          zeitpunkt : Integer
53      end
```

Assoziationen

```
1  --assoziationen
```

```
2     association MapTiles between -- Status als Beziehung (Qualifier)
          über Messwert (PAG) - [Status] -> (Druck))
3         PAGeraet[1] role geraet qualifier (x:Status)
4         Messwert[0..1] role wert -- Felder
5     end
6
7     association A_Einsatzkraft_PAGeraet between
8         Einsatzkraft[1]
9         PAGeraet[1]
10    end
11
12    association A_Trupp_Einsatzauftrag between
13        Trupp[1]
14        Einsatzauftrag[1] role aktuellerEinsatzauftrag
15    end
16
17    association A_Trupp_Einsatzkraft between
18        Trupp[1]
19        Einsatzkraft[2..3] role Einsatzkraefte ordered
20    end
21
22    association A_Trupp_Einsatzinformationen between
23        Einsatzinformationen[1]
24        Trupp[3]
25    end
```

Invarianten und Pre/Postconditions

```
1  --invarianten
2  constraints
3      context Einsatzinformationen
4          inv definiereGrundlegendeEinsatzinformationen: --ocl1
5              self.EinsatzstrasseundHausnummer.isDefined() and
6              self.Einsatznummer.isDefined() and
7              self.Rufgruppe.isDefined()
8
9      context Trupp
10         inv funkrufnameTrupp: --ocl2
11             self.Funkrufname.isDefined()
12         inv nurBestimmteEinsatzstatus: --ocl3
13             (self.Einsatzstatus = Status::NICHTS or
```

```
14              self.Einsatzstatus = Status::START or
15              self.Einsatzstatus = Status::ENDE)
16          inv beiEinsatzbeginnCountdownDefiniert: --ocl4
17              (self.Einsatzstatus = Status::NICHTS implies self.Timer.
                    isUndefined()) and
18              (self.Einsatzstatus = Status::START implies self.Timer.
                    isDefined()) and
19              (self.Einsatzstatus = Status::ENDE implies self.Timer.
                    isDefined())
20          inv EinsatzkraefteMitRichtigerQualifikation: --ocl5
21              self.Einsatzkraefte->size() = 2 implies (self.
                    Einsatzkraefte->exists(ek | ek.Funktion = Funktion::
                    TRUPPFUEHRER) and self.Einsatzkraefte->exists(ek | ek.
                    Funktion = Funktion::TRUPPMANN)) and
22              self.Einsatzkraefte->size() = 3 implies (self.
                    Einsatzkraefte->exists(ek | ek.Funktion = Funktion::
                    TRUPPFUEHRER) and self.Einsatzkraefte->exists(ek | ek.
                    Funktion = Funktion::TRUPPMANN) and self.
                    Einsatzkraefte->exists(ek | ek.Funktion = Funktion::
                    TRUPPMANN2))
23
24      context Einsatzauftrag
25          inv definiereEinsatzauftrag: --ocl6
26              self.Auftrag.isDefined()
27
28      context Einsatzkraft
29          inv definiereGrundlegendeEinsatzkraftinformationen: --ocl7
30              self.Name.isDefined() and
31              self.Funktion.isDefined()
32
33      context PAGeraet
34          inv definiereGrundlegendePAInformationen: --ocl8
35              self.maximalerDruck >= 0
```

B Anforderungen an die Applikation

B.1 Funktionale Anforderungen

Im folgenden Abschnitt werden die funktionalen Anforderungen an die Software gestellt:

B.1.1 Erfassen von Einsatzinformationen

Der Anwender muss grundelegende Informationen zum Einsatz, wie Datum, Einsatzort, Einheitsführer und Rufgruppe erfassen können.

B.1.2 Erfassen von Trupps

Der Anwender muss verschiedene Trupps mit Truppnamen, Name der Mitglieder des Trupps, Einsatzauftrag und Verortung, Funkrufnamen und Behälterdrücken erfassen können.

B.1.3 Uhrzeit bei Luftdruckangabe

Das System muss automatisch die Uhrzeit erfassen, bei der der Luftdruck gemessen worden ist.

B.1.4 Ermittlung des Rückzugsolldrucks

Das System muss automatisch errechnen, wann der Trupp sich spätestens nach Erreichen des Ziels zurückziehen muss.

B.1.5 Speichern des Systemstands

Der Anwender muss den aktuellen Einsatzstand und alle erfassten und kalkulierten Daten speichern können, sodass diese erneut geladen werden können.

B.1.6 Laden des Systemstands

Der Anwender muss den letzten gespeicherten Einsatzstand und alle erfassten und kalkulierten Daten laden können, sodass der Stand des gespeicherten Einsatzes wiederhergestellt ist.

B.1.7 Periodische Speicherung des Einsatzstands

Das System muss in bestimmten Zeitintervallen den aktuellen Einsatzstand speichern, damit dieser bei Fehlern oder Systemabstürzen wiederhergestellt werden kann.

B.1.8 Einsatzzeiterfassung

Der Anwender muss die Möglichkeit besitzen einen Timer für jeden Trupp zu stellen. Die Timerzeit entsprechend der maximalen Einsatzzeit soll individuell, aber nach realisitischen Einsatzzeiten unter Atemschutz einstellbar sein.

B.1.9 Mitteilung bei Ein-Dritteln der Einsatzzeit

Der Anwender muss bei Überschreiten von Ein-Dritteln seiner eingestellten Einsatzzeit darüber informiert werden, dass eine Druckabfrage erforderlich ist.

B.1.10 Mitteilung bei Zwei-Dritteln der Einsatzzeit

Der Anwender muss bei Überschreiten von Zwei-Dritteln seiner eingestellten Einsatzzeit darüber informiert werden, dass eine Druckabfrage erforderlich ist.

B.1.11 Visuelle Darstellung der Zeiterfassung

Der Anwender muss kontinuierlich über die Dauer der aktuellen Einsätze der Trupps, die verbleibende Einsatzdauer der Trupps sowie die verbleibenden Intervalle von einem Drittel und zwei Dritteln der Einsatzzeit informiert werden können.

B.1.12 Möglichkeit zur Beendigung der Zeiterfassung

Der Anwender muss die Möglichkeit besitzen, den Einsatz vorzeitig zu beenden und somit alle Timer anzuhalten.

B.1.13 Überschreiten der maximalen Einsatzzeit

Der Anwender muss unignorierbar darüber gewarnt werden, wenn die maximale Einsatzzeit eines Trupps überschritten ist.

B.2 Nicht-funktionale Anforderungen

Im folgenden Abschnitt werden die nicht-funktionalen Anforderungen an die Software gestellt:

B.2.1 Benutzerfreundlichkeit

Die Applikation sollte einfach aufgebaut und intuitiv bedienbar sein.

B.2.2 Schutz vor Fehlbedienungen

Die Applikation sollte bei folgenschweren Funktionen Schutzmechnanismen vorweisen, die die fehlerhafte Bedienung erschweren.

B.2.3 Fehlerhafte Nutzereingaben

Die Möglichkeit fehlerhafte Eingaben zu tätigen sollte soweit wie möglich reduziert werden.

B.2.4 Zuverlässigkeit

Die Applikation sollte zu jeder Zeit funktionieren und im Fehlerfall den vorherigen Stand wiederherstellen können.

C Verwendete Design-Patterns

C.1 Dependency Injection

```
1  public class InjectorManager {
2      public static InjectorManager IM;
3      public Einsatzinformationen einsatzinformationen;
4      public Ueberwachungstafel ueberwachungstafel;
5      public Datenverwaltung datenverwaltung;
6      public UeberwachungstafelAdapter ueberwachungstafelAdapter;
7      boolean neuerEinsatzIntent = false;
8
9      public InjectorManager() {
10         IM = this;
11     }
12
13     public Datenverwaltung gibDatenverwaltung() {
14         return datenverwaltung;
15     }
16
17     public void setzeDatenverwaltung(Datenverwaltung datenverwaltung)
           {
18         this.datenverwaltung = datenverwaltung;
19     }
20
21     //...
22 }
```

C.2 Adapter-Pattern

Dieser Adapter wurde zum Testen gebaut und extrahiert die Klassen von dem direkten Zugriff auf das Layout

```
1  public class UeberwachungstafelAdapter {
2      private Ueberwachungstafel ueberwachungstafel;
3
4      public UeberwachungstafelAdapter() {
5          if(InjectorManager.IM != null)
6          {
7              this.ueberwachungstafel = InjectorManager.IM.
                   gibUeberwachungstafel();
8          }
9      }
10
11     public void onTick(int zeit, Trupp trupp) {
12         if (ueberwachungstafel != null) ueberwachungstafel.onTick(
               zeit, trupp);
13     }
14 //...
15 }
```

D Verwendete Testpattern

D.1 Arrange Act Assert

Klar strukturierter Testfall durch Anwendung des AAA-Pattern.

```
1  @Test
2  public void testSetEinsatzstatusgültig() {
3      // Arrange
4      Status expectedStatus = Status.START;
5      Trupp trupp = new Trupp("Trupp 1");
6
7      // Act
8      trupp.set_einsatzstatus(expectedStatus);
9      Status actualStatus = trupp.get_einsatzstatus();
10
11     // Assert
12     Assert.assertEquals(expectedStatus, actualStatus);
13 }
```

D.2 Äquivalenzklassentest

Auszug aus den definierten OCL-Constraints unter A

```
1 context PAGeraet
2     inv definiereGrundlegendePAInformationen: --ocl8
3         self.maximalerDruck >= 0
```

Daraus lassen sich folgende Äquivalenzklassen (gÄq, gültig; uÄq, ungültig) ableiten:

$$g\ddot{A}q = \{x | x \geq 0\} \tag{D.1}$$

$$u\ddot{A}q = \{x|x < 0\} \tag{D.2}$$

Bei der Auswahl von Beispielhaften Werten wurde sich für die 100, 0 und -100 entschieden. Die 0 stellt einen Grenzfall dar, weshalb diese im folgenden auch getestet wird:

```
@Test
public void testPAGeraet() throws Exception {
    // Äquivalenzklasse 1: Druck ist positiv oder null
    int positiverDruck = 100;
    PAGeraet paGeraet1 = new PAGeraet(positiverDruck);
    assertEquals(positiverDruck, paGeraet1.getDruck(Status.START));

    int NullDruck = 0;
    PAGeraet paGeraet2 = new PAGeraet(NullDruck);
    assertEquals(NullDruck, paGeraet2.getDruck(Status.START));

    // Äquivalenzklasse 2: Druck ist negativ
    int negativerDruck = -100;
    assertThrows(IllegalArgumentException.class, () -> {
        new PAGeraet(negativerDruck);
    });
}
```

D.3 Manueller Systemtest

Auf Grundlage des Prozessmodells 3.5 wurde folgender Systemtest entwickelt und durchgeführt:

```
Manueller Systemtest:
Erfassen von Einsatzinformationen:
Einsatznummer: ENR123456
Adresse: Teststrasse 1
Rufgruppe: DMO 05
Einheitsführer: Mayer

1 Trupp: Angriffstrupp 33-HLF-1
Maximale Einsatzzeit: 30min
Einsatzauftrag: 2OG, rechts, löschen
Angriffstruppführer: Schröder
```

```
  - Startdruck: 280bar
Angriffstruppmann: Möller
  -Startdruck: 300bar

Erste Rückmeldung nach 4min seit Einsatzbeginn:
  Ziel erreicht!
  Druck Schröder 240bar
  Druck Möller 250bar

Zweite Rückmeldung nach 9min seit Einsatzbeginn:
  Kontrolle 1/3-Zeit:
  Druck Schröder 200bar
  Druck Möller 180bar

Dritte Rückmeldung nach 12min seit Einsatzbeginn:
  Trupp zieht sich zurück!
  Druck Schröder 150bar
  Druck Möller 100bar

Vierte Rückmeldung nach 15min seit Einsatzbeginn:
  Trupp hat EST verlassen. Trupp legt ab
  Druck Schröder 100bar
  Druck Möller 50bar

2 Trupp: Wassertrupp 33-HLF-1
Maximale Einsatzzeit: 30min
Einsatzauftrag: 3OG, links, löschen
Angriffstruppführer: Neubert
  - Startdruck: 300bar
Angriffstruppmann: Reubert
  -Startdruck: 310bar


Zweite Rückmeldung nach 9min seit Einsatzbeginn:
  Kontrolle 1/3-Zeit:
  Druck Schröder 200bar
  Druck Möller 180bar

Dritte Rückmeldung nach 12min seit Einsatzbeginn:
  Trupp zieht sich zurück!
```

```
52   Druck Schröder 150bar
53   Druck Möller 100bar
54
55 Vierte Rückmeldung nach 15min seit Einsatzbeginn:
56   Trupp hat EST verlassen. Trupp legt ab
57   Druck Schröder 100bar
58   Druck Möller 50bar
59
60
61 3 Trupp: Schlauchtrupp 33-HLF-1
62 Maximale Einsatzzeit: 30min
63 Einsatzauftrag: 2OG, rechts, löschen
64 Truppführer: Müller
65   - Startdruck: 290bar
66 Truppmann: Schmidt
67   - Startdruck: 280bar
68
69 Zweite Rückmeldung nach 9min seit Einsatzbeginn:
70   Kontrolle 1/3-Zeit:
71   Druck Müller 250bar
72   Druck Schmidt 240bar
73
74 Dritte Rückmeldung nach 12min seit Einsatzbeginn:
75   Trupp zieht sich zurück!
76   Druck Müller 220bar
77   Druck Schmidt 200bar
78
79 Vierte Rückmeldung nach 15min seit Einsatzbeginn:
80   Trupp hat EST verlassen. Trupp legt ab
81   Druck Müller 180bar
82   Druck Schmidt 150bar
```

E Codeanalyse im Hinblick auf Boilerplate-Code zwischen Java und Kotlin

```java
public class Einsatzauftrag {
    String auftrag;
    int maximaleEinsatzzeit;
    String bemerkungen;

    public Einsatzauftrag(String auftrag, int maximaleEinsatzzeit,
        String bemerkungen) {
        if (maximaleEinsatzzeit < 0)
            throw new IllegalArgumentException("Die maximale
                Einsatzzeit darf nicht negativ sein.");
        if (auftrag == null)
            throw new IllegalArgumentException("Der Auftrag darf
                nicht null sein.");
        this.auftrag = auftrag;
        this.maximaleEinsatzzeit = maximaleEinsatzzeit;
        this.bemerkungen = bemerkungen;
    }
    //Getter, Setter & weitere Methoden....
}
```

Hier sehen wir einen Codeausschnitt der Klasse Einsatzauftrag der nur die Deklaration der Variablen so wie des Konstruktors zeigt. Der Code umfasst 16 LOC.

```
1 class Einsatzauftrag(val auftrag: String, val maximaleEinsatzzeit:
      Int, val bemerkungen: String) {
2     init {
3         require(maximaleEinsatzzeit >= 0) { "Die maximale Einsatzzeit
              darf nicht negativ sein." }
4     }
5     //Getter, Setter & weitere Methoden....
6 }
```

Hier sehen wir den Code in der Programmiersprache Kotlin. Eine implizte Konstruktordeklaration mit allen Variablen als Parametern erfolgt bereits in der Programmiersprache Kotlin automatisch. Eine Nullsicherheitsfunktion in der Programmiersprache Kotlin macht hier das explizite Abfragen nach null unnötig.
Der vergleichbare Code umfasst nur 6 LOC.

Glossar

Access To Foreign Data Anzahl der direkten Zugriffe einer Klasse auf die Attribute anderer Klassen.

Direct Mode Operation DMO ist der netzunabhängige Betrieb von BOS-Digitalfunkgeräten. Es basiert auf dem Master/Slave-Modell. Es sind Einzelrufe und Gruppenrufe in Halb-Duplex möglich.

Einheitsführer Ein Einheitsführer übernimmt die Rolle als Leiter seiner taktischen Einheit. [8, S.15] Je nachdem, ob ein weiterer höhergestellter Führungsdienst anwesend ist, kann dieser auch als Gesamteinsatzleiter fungieren. Der Einheitsführer trifft die Entscheidungen für seine Einheit und befehligt seine Teileinheiten. Obwohl er unter bestimmten Bedingungen von den Dienstvorschriften abweichen darf [8, S.6] [7, S.32], ist er stets zur Rechenschaft verpflichtet und trägt die Verantwortung für seine Entscheidungen [6, S.13].

Mockito Mockito ist ein Open-Source-Framework für Java, das in der Softwareentwicklung zum Testen von Code verwendet wird. Es ermöglicht das Erstellen von Mock-Objekten, die als Platzhalter für reale Objekte dienen und das Verhalten anderer Objekte simulieren können.

Taktische Einheit Die taktische Einheit ist der Begriff für eine Einheit von Mannschaft und Gerät. Dabei besteht eine taktische Einheit immer aus einem Einheitsführer, einem Maschinisten und mindestens einem Trupp. [8, S.7] Untergliedert werden die Einheiten in verschiedene Mannschaftsstärken, wobei der selbstständige Trupp (Truppführer, Truppmann und Maschinist) die kleinste taktische Einheit bildet.

Tight Class Cohesion Anzahl der direkt gekoppelten public-Methoden einer Klasse geteilt durch die maximale Anzahl der Verbindungen der Methoden.

Trunked Mode Operation TMO ist der Modus für BOS-Digitalfunkgeräte unter Verwendung der Netzinfrastruktur. Es sind Einzelrufe (Halb-Duplex oder Voll-Duplex) als auch Gruppenrufe (Halb-Duplex) möglich.

Trupp Ein unselbstständiger Trupp ist ab der taktischen Einheit Staffel, als Teil der Mannschaft vorhanden. Dieser Trupp besteht aus zwei Truppmitgliedern (Truppführer und Truppmann). [8, S.39] Im Spezialfall (besondere Einsatzstellen insbesondere unter Atemschutz) kann dieser um ein weiteres Truppmitglied ergänzt werden. [7, S.17] Ein unselbstständiger Trupp wird von einem Truppführer geführt und ist direkt unter dem Einheitsführer seiner jeweiligen taktischen Einheit unterstellt. [8, S.15] Ein selbstständiger Trupp ist hingegen bestehend aus mindestens einem Maschinisten, einem Truppführer mit der Qualifikation eines Gruppenführeres, einem Truppmann und Gerät und bildet eine eigenständige taktische Einheit.

Weighted Method Count Summe der zyklomatischen Komplexität aller Methoden einer Klasse.

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original