

**Terminal-Config-Manager:
Projektarbeit zur Ausbildung zum
Informatiker für Anwendungsentwicklung
CHECK24 Tech Hub und Services GmbH**

Adrian Schurz

II. November 2022

1 Projektantrag

Diese Seite wird später entfernt.

1 Projektantrag

Der folgende Projektantrag wurde um die Auflagen, welche in der Terminbestätigung genannt wurden, erweitert. Weiterhin hat sich der Name der Firma, ohne jegliche Änderungen des Arbeitsverhältnisses, in der Zwischenzeit geändert und wurde hier aktualisiert.

Abschlussprüfung im Beruf Fachinformatiker für Anwendungsentwicklung

Winter 2022/23

Antragsformular

Azubi-Nr.: 480513

Name: Adrian Schurz

Ausbildungsbetrieb / Praktikumsbetrieb	CHECK24 Tech Hub und Services GmbH
Projektbezeichnung	terminal-config-manager
Projektbeschreibung	<p>Motivation: Bei der Arbeit an Software-Anwendungen, welche eng mit einer Vielzahl anderer solcher Anwendungen über mehrere Umgebungen hinweg interagieren, ergaben sich zwei häufige Anwendungsfälle beim Umgang mit Konfigurationsdateien.</p> <ol style="list-style-type: none">1. Das Einsehen und Kontrollieren von Einträgen innerhalb dieser Dateien. Dabei gibt es eine große Zahl an über verschiedene Verzeichnisse verstreuten Dateien. Innerhalb einer Datei gibt es jeweils eine große Zahl an Einträgen, von denen aber oft nur wenige relevant sind.2. Das Ändern eines solchen Eintrags, um das Verhalten der zugehörigen Anwendung anzupassen. <p>Sowohl das Kontrollieren als auch das Ändern von beliebigen Einträgen soll schnell und in einer Anwendung möglich sein.</p> <p>Beispielszenario</p> <p>In der Continuous-Integration-Pipeline (CI) einer zentralen Anwendung schlagen unvermittelt Akzeptanztests fehl. Um unverzüglich die Ursache feststellen zu können, muss das lokale Setup, das in diesem Moment an die Entwicklung einer anderen Anwendung und für die Ausführung einer anderen Menge an Tests konfiguriert ist, umkonfiguriert werden.</p> <p>IST-Zustand:</p> <p>Um die Zielanwendung zu untersuchen sind viele Teilaufgaben zu erledigen, beispielsweise das Aktivieren detaillierter</p>

	<p>Lognachrichten, das Ändern der auszuführenden Unit-, Integration- und Systemtestsuites, das Ändern von Cache-Verhalten, das anpassen von Ziel-IPs anderer, an der Testausführung beteiligter Anwendungen. Jeder dieser Schritte ist wiederum mit mehreren Einzelschritten verbunden. Meist muss zu diesem Zweck eine passende IDE gestartet, die relevante Konfigurationsdatei ermittelt und zum richtigen Unterordner navigiert werden. Danach muss die Datei geöffnet, ihr Inhalt untersucht und der Zielwert geprüft und gegebenenfalls angepasst werden. In Summe sind sehr viele, repetitive Einzelaufgaben händisch zu erledigen. Hier bietet sich daher Potential zur Verbesserung.</p> <p>Soll-Zustand:</p> <p>Es existiert ein Kommandozeilenprogramm, das übersichtlich eine Liste anzeigt. Die Liste enthält pro Zeile einen Beschreibungstext und den aktuellen Wert innerhalb der Zieldatei. Das Programm erlaubt es, per Tastendruck Zeilen auszuwählen und den zugehörigen Wert aus einer Menge möglicher Werte auszuwählen. Wird ein neuer Wert ausgewählt, so wird die dazugehörige Konfigurationsdatei umgeschrieben. Sowohl die Zieldateien als auch die jeweiligen möglichen Werte sind konfigurierbar. Zusätzlich existieren ausführliche Unit- und Akzeptanztests.</p>		
Projektumfeld	<p>Die CHECK24 Vergleichsportal GmbH, CHECK24 Vergleichsportal Reise GmbH und CHECK24 Tech Hub und Services GmbH sind Betreiber von check24.de, einer Website, auf der verschiedene Produkte zum Vergleich angeboten werden. Der Auftraggeber CHECK24 Tech Hub und Services GmbH betreibt auf dieser Online-Plattform eine Vergleichsmöglichkeit von Pauschalreisen.</p> <p>Die Menge und Komplexität interner Anwendungen, die am Produktionsbetrieb und der Qualitätssicherung beteiligt sind, wächst stetig. Damit einher gehen komplexere Interaktionen und Konfigurationsmöglichkeiten, die häufige Fehlerquellen im Betrieb und während der Entwicklung darstellen. Je einfacher diese Konfiguration möglich ist, desto schneller kann während der Fehlersuche und der Entwicklung gearbeitet werden.</p>		
Projektphasen (einschließlich Zeitplanung)	<table border="1"> <thead> <tr> <th>Phase</th><th>Dauer (h)</th></tr> </thead> </table>	Phase	Dauer (h)
Phase	Dauer (h)		

	Konzeption	10
	Wahl des Techstacks	3
	Einrichtung Entwicklungsumgebung	2
	Implementierung	30
	Qualitätssicherung	15
	Dokumentation	10
	Gesamt	70
Dokumentation zur Projektarbeit (nicht selbstständig erstellte Dokumente sind zu kennzeichnen)	<p>Das Projekt wird ausführlich dokumentiert. Dazu gehören:</p> <ul style="list-style-type: none"> • Eine ausführliche, aus Quellcodekommentaren generierte Dokumentation der einzelnen Programmmodule im HTML-Format • Hilfetexte, die vom Programm selbst bei Bedarf ausgegeben werden • README-Dateien, welche das Aufsetzen des Projekts, die Kompilierung und Ausführung der Unit- und Akzeptanztests beschreiben • Eine Projektdokumentation im PDF-Format, welches eine IST-Analyse, die Anforderungen an die Software, die Projektziele, die Zeitplanung, den Projektverlauf und das Pflichtenheft beinhaltet. • Die Planung des Projekts • Die Umsetzung des Projekts • Eine Ergebnisdiskussion 	
Bearbeitungsdauer von	1.10.2022	
Bearbeitungsdauer bis	14.11.2022	
Präsentationsmittel	Laptop	
Overheadprojektor	vorhanden	
Projektionsbildschirm (als Beamer nutzbar)	vorhanden	
Andere Präsentationsmittel (sind vom Prüfling mitzubringen)	Laptop	

Themenbetreuer	Jessica Parth Falk Döring
----------------	------------------------------

Themenbestätigung

Folgendes wurde aus Ihrem Schreiben vom 23.09.2022, der Zustimmung des betrieblichen Auftrages mit Auflagen, übernommen:

Thema bestätigt	
Mit Auflagen bestätigt	X Projektdokumentation umfasst auch: Planung, Umsetzung, Ergebnisdokumentation -> Planung entsprechend anpassen
Grund Ablehnung	

2 Nachweisblatt

Diese Seite wird später entfernt.

Abschlussprüfung IT-Berufe: Nachweis für den betrieblichen Auftrag

- | | |
|---|---|
| <input type="checkbox"/> IT-Systemelektroniker/-in | <input type="checkbox"/> Systemkaufmann/-frau |
| <input type="checkbox"/> Informatikkaufmann/-frau | <input type="checkbox"/> Fachinformatiker/-in Anwendungsentwicklung |
| <input type="checkbox"/> Fachinformatiker/-in Systemintegration | |

Name, Vorname:

Prüfungsnummer:

Datum:

Zeitraum (Stunden):

Unterschrift
Prüfungsteilnehmer:

Unterschrift
Themenbetreuer:

Ich versichere, dass ich den betrieblichen Projektauftrag einschließlich Dokumentation selbstständig und nur mit den angegebenen Hilfsmitteln erstellt habe.

Ort, Datum:

Unterschrift Prüfungsteilnehmer:

Ort, Datum:

Unterschrift Themenbetreuer:

Datenschutz:

Die IHK Dresden ist für die Durchführung von Prüfungen in der Aus- und Weiterbildung zuständig. Die Ermächtigung zur Datenverarbeitung in diesem Zusammenhang ergibt sich aus Art. 6 Abs.1 Buchstabe e DSGVO.

Hinweis: Für Prüfungsergebnisse und Unterlagen ergeben sich zum Teil vom Üblichen abweichende Aufbewahrungsfristen.

Prüfungsergebnisse aus der beruflichen Bildung werden 50 Jahre aufbewahrt, da über die Zeit des gesamten Erwerbslebens die Möglichkeit der Ausstellung einer Zeugnisweitschrift gewahrt werden muss.

Prüfungsunterlagen werden hingegen ein Jahr nach Erlangen der rechtlichen Bestandskraft des Ergebnisses vernichtet.

Sie können Widerspruch gegen die Verarbeitung einlegen (Art. 21 DSGVO). Sollten Sie davon Gebrauch machen, prüft die IHK, ob die gesetzlichen Voraussetzungen hierfür erfüllt sind. Hinweis: Die zur Erfüllung der hoheitlichen Aufgaben notwendigen Daten können in der Regel nicht vor Ablauf der Speicherfrist gelöscht werden.

Die umfassende Datenschutzerklärung der IHK Dresden finden Sie unter <https://www.dresden.ihk.de/datenschutz>. Den Widerspruch können Sie durch Nutzung des [Widerspruchsformulars](#) auf der Webseite, schriftlich bei der IHK widerspruch@dresden.ihk.de einlegen.

Ort, Datum:

Unterschrift Prüfling:

Ort, Datum:

Unterschrift Themenbetreuer:

Inhaltsverzeichnis

1 Projektantrag

2 Nachweisblatt

3 Analyse 1

3.1 IST-Zustand 1

3.2 Soll-Zustand 2

4 Technologie 2

4.1 Kriterien 2

4.2 Auswahl 3

5 Projektplanung 3

5.1 Zeit 3

5.2 Ressourcen 4

5.3 Design 4

5.4 Projektorganisation 6

6 Umsetzung 6

6.1 Einrichtung 6

6.2 Entwicklung 7

6.3 Dokumentation 7

6.4 Qualitätssicherung 8

7 Ergebnisdiskussion 10

7.1 Funktionalität 10

7.2 Domain Driven Design 10

7.3 Moduldokumentation 10

7.4 Testabdeckung 11

7.5 Bekannte Fehler 11

8 Anhang 12

9 Kundendokumentation 17

9.1 Beschreibung 17

9.2 Installation 17

9.3 Konfiguration 18

9.4	Benutzung	21
9.5	Problembehandlung	22
Literatur		24
Glossar		26

3 Analyse

Der Bedarf für die im Rahmen dieser Projektarbeit erstellte Softwarelösung ergab sich bei der Arbeit an Software-Anwendungen, welche eng mit einer Vielzahl anderer Anwendungen und über mehrere Umgebungen hinweg interagieren.

3.1 IST-Zustand

Der Großteil dieser Anwendungen besitzt weitläufige Konfigurationsmöglichkeiten welche ihren Betrieb in verschiedensten Szenarien steuern. Beispiele für Konfigurationsmöglichkeiten und deren Ausprägungen sind:

- Das Loggingverhalten der Anwendung
 - Logging gegen die Logverarbeitungssoftware der Produktionsumgebung
 - Logging gegen eine lokale Instanz der Logverarbeitungssoftware
 - Logging auf das Dateisystem
 - Logging mit verschiedenen Logleveln
- Die Zieldatenbank der Anwendung
 - Datenbank der Produktionsumgebung
 - Datenbank der Testumgebung
 - lokale Datenbank
- Die Ausführung von Softwaretests
 - Ausführen von ausschließlich Unittests
 - Ausführen von Akzeptanztests
 - Ausführen der Gesamtheit der Tests
 - Anpassung der Ziel-IP einer weiteren, für die Testausführung notwendigen Anwendung

Der Kontext der Arbeit an der Software wechselt regelmäßig zwischen Entwicklung und der Behandlung von Fehlern, welche im Produktions- oder Testbetrieb auftreten. Um dabei das beobachtete Verhalten der Anwendung korrekt zu interpretieren sind u.a. zwei Arbeitsschritte häufig zu erledigen:

- **Prüfen** der aktuellen Konfiguration
- **Anpassen** der aktuellen Konfiguration

Die Anzahl der an jedem Einzelfall beteiligten Anwendungen und die Anzahl der Konfigurationsdateien pro Anwendungen führen dazu, dass jeweils viele verschiedene und weit über das Dateisystem verstreute Dateien relevant sind. Pro Datei und Einzelfall sind folgende Arbeitsschritte zu erledigen:

- Starten der zur Anwendung gehörigen IDE
- Ermitteln der Konfigurationsdatei
- Navigation im Verzeichnisbaum
- Öffnen der Datei
- Finden des relevanten Eintrags in einer mitunter sehr langen Textdatei
- Ermitteln der möglichen Zielwerte
- Ändern des Eintrags
- Speichern der Datei

Diese Einzelschritte, multipliziert mit der Anzahl an Dateien, stellen eine große Menge an repetitiven Handlungen dar. Wenn diese erleichtert würden, dann ließen sich sowohl Zeit und Konzentrationsvermögen einsparen als auch Fehlerpotential verringern.

3.2 Soll-Zustand

Es existiert ein Kommandozeilenprogramm, das übersichtlich eine Liste anzeigt. Die Liste enthält pro Zeile einen Beschreibungstext und den aktuellen Wert innerhalb der Zielfeld. Das Programm erlaubt es, per Tastendruck Zeilen auszuwählen und den zugehörigen Wert aus einer Menge möglicher Werte auszuwählen. Wird ein neuer Wert ausgewählt, so wird die dazugehörige Konfigurationsdatei zum neuen Wert hin umgeschrieben. Sowohl die Zielfeld als auch die jeweiligen möglichen Werte sind konfigurierbar. Zusätzlich existieren ausführliche Unit- und Akzeptanztests.

4 Technologie

4.1 Kriterien

Für dieses Projekt bieten sich grundsätzlich alle gängigen Programmiersprachen an. Während der Umsetzung soll ausgewählten Aspekten der Softwareentwicklung gesonderte Aufmerksamkeit zukommen.

Korrektheit und Laufzeitstabilität Es soll auf technischem Weg zum Einen sichergestellt werden, dass sich das Programm zu jedem Zeitpunkt erwartungsgemäß und korrekt verhält und zum Anderen, dass Fehlerzustände zur Laufzeit so weit wie möglich ausgeschlossen werden.

Als hauptsächliche Wege dies zu erreichen werden folgende Ansätze gewählt:

- Wahl einer Programmiersprache mit strenger Typisierung
- Wahl einer kompilierten Programmiersprache mit vergleichsweise starken Garantien zum Laufzeitverhalten
- Einbeziehung von Property-Based-Testing [17] in das Konzept der Softwaretests

Ausführliche, vom Quellcode abgeleitete Moduldokumentation Neben der Projektdokumentation soll eine Dokumentation der einzelnen Softwaremodule entstehen. Um dem Problem zu begegnen, dass Dokumentation und Quellcode im Laufe der Entwicklung auseinanderlaufen soll die Moduldokumentation direkt aus der Quellcodestruktur und den Quellcodekommentaren generierbar sein.

4.2 Auswahl

Als Programmiersprache und Build-System wurden auf Basis der obengenannten Ziele Haskell [16] und Stack [22] gewählt.

Stack bietet neben seiner Hauptaufgabe die Software-Abhängigkeiten des Projekts zu verwalten und den Buildvorgang zu steuern die Möglichkeit Moduldokumentation im HTML-Format anhand der Quellcodestruktur und der Quellcodekommentare generieren während Haskell eine typischere, kompilierte Programmiersprache mit Unterstützung für Property-Based-Testing darstellt.

5 Projektplanung

5.1 Zeit

Die geplante Umsetzungsdauer beträgt ca. 70 Stunden und ist folgendermaßen gegliedert.

Projektphase	geplante Zeit (h)
Konzeption	10
Technologiewahl	3
Einrichtung	2
Implementierung	30
Qualitätssicherung	15
Dokumentation	10
Gesamt	70

5.2 Ressourcen

Im Folgenden werden alle zur Fertigstellung des Projekts verwendeten Ressourcen, sowohl Hardware- als auch Software- und Personalressourcen, aufgelistet.

- Hardware
 - Lenovo Thinkpad P52
- Software
 - Arch Linux (Betriebssystem) [3]
 - Stack (Buildsystem) [22]
 - GHC (Compiler) [13]
 - Git (Versionskontrolle) [10]
 - Haddock (Generierung der Moduldokumentation) [15]
 - Graphmod (Graphgenerierung, Modulabhängigkeiten) [14]
 - Visual Studio Code (Codeeditor) [25]
 - T_EX-Live (Projektdokumentation) [1]
- Personal
 - ein Softwareentwickler zur Umsetzung
 - ein Softwareentwickler zur Qualitätskontrolle

5.3 Design

Nutzerinteraktion Für die Art der Benutzerinteraktion des Programms sind mehrere Ansätze denkbar. Beispielsweise könnte das Programm mit jeweils verschiedenen, aufeinanderfolgenden Befehlen von der Kommandozeile aus aufgerufen werden wie in Abb. 1 dargestellt. Alternativ könnte das Programm, wie in Abb. 2 skizziert, mit einer kompletten, grafischen Oberfläche versehen werden.

Abbildung 1: Nutzerinteraktion mit aufeinanderfolgenden Befehlen

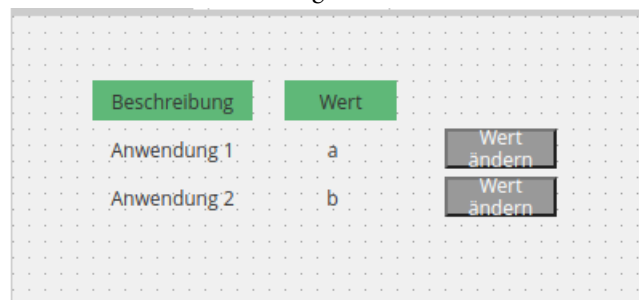
```
>_ terminal-config-manager show

1 Anwendung a mit Wert -> b
2 Anwendung c mit Wert -> d

>_ terminal-config-manager change 2

1 Anwendung a mit Wert -> b
2 Anwendung c mit Wert -> e
```

Abbildung 2: Nutzerinteraktion über eine grafische Oberfläche und Buttons, skizzenhaft



Als pragmatischer, einfacher und funktionaler Ansatz wurde ein Mittelweg gewählt bei dem eine Konsolenanwendung beim Start alle einzelnen konfigurierten Zielwerte auf jeweils einer Zeile darstellt, geöffnet bleibt und auf Tasteneingaben wartet um entweder eine seiner Funktionen auszuführen oder beendet zu werden (siehe Abb. 14 im Kapitel 9 - Kundendokumentation).

Software design Jeweils funktional zusammengehörige Teile des Quellcodes wurden so gut wie möglich von anderen Teilen separiert. Die hauptsächliche Organisationsebene bilden dabei Module, die sich in jeweils einer Quellcodedatei befinden. Beispielsweise ist dadurch Programmcode, der mit der Verarbeitung von Tasteneingaben betraut ist, im Modul

```
src/UserInterface/Input.hs
```

lokalisiert. Programmcode, welcher mit dem Dateisystem interagiert, dagegen hier:

```
src/Infrastructure/FileModification.hs
```

Den einzelnen Modulen übergeordnet befindet sich eine Organisationsebene, welche sich an einem möglichen Ansatz aus dem Kontext des Domain-Driven-Design [9] orientiert. Dabei werden Teile

des Programmcode in verschiedene Gruppen organisiert, beispielsweise **Application**, **Domain**, **Infrastructure** und **Userinterface**, und es wird darauf geachtet, dass Abhängigkeiten zwischen Modulen dieser Gruppen stets in eine bestimmte Richtung zeigen. Wenn dieses Schema eingehalten wird, dann können Probleme, wie zyklische Abhängigkeiten oder schlecht erweiterbare Module, reduziert oder ganz vermieden werden. Das angestrebte Schema ist dem Buch Domain-Driven-Design [9] entnommen und ist in Abb. 9 im Angang dargestellt. Zum Vergleich wurden die tatsächlichen Modulabhängigkeiten mit Unterstützung von Softwaretools [14] [29] ausgelesen und in einem Graphen dargestellt. In Kapitel 7.2 - Ergebnisdiskussion wird ein Vergleich angestellt.

5.4 Projektorganisation

Die Hauptelemente der Verzeichnisstruktur des Projekts sind wie in Abb. 12 im Anhang dargestellt organisiert. Die Verzeichnisse enthalten dabei in der angezeigten Reihenfolge

- `src`: Quellcode des Programms
- `test`: Quellcode der Unit- und Integrationstests
- `doc`: generierte Moduldokumentation und Projektdokumentation
- `distribution`: Skripte um Softwarepakete für verschiedene Betriebssysteme zu erstellen
- `bin`: kompilierte, ausführbare Dateien
- `tool`: Hilfsskripte, z.B. zur Generierung des Modulabhängigkeitsgraph

Die Datei `package.yaml` enthält die für Stack notwendigen Metadaten, beispielsweise Deklarationen der verwendeten Softwarebibliotheken und Konfiguration des Buildprozesses.

Die Datei `README.md` enthält eine kurze Beschreibung des Programms und eine Nutzungsanleitung zusammen mit offenen Aufgaben und Zusatzinformationen.

6 Umsetzung

6.1 Einrichtung

Es wurde der zur Entwicklung im Tagesgeschäft bei Check24 bereitgestellte Laptop für das Projekt genutzt auf dem bereits ein Linux-Betriebssystem installiert war. Auch die genutzte IDE war bereits aufgrund anderer Projekte vorinstalliert. Lediglich die \LaTeX -Entwicklungsumgebung und Details der Build-Umgebung mussten speziell für dieses Projekt konfiguriert werden.

Abbildung 3: Beispiel eines Modul-Codekommentars

```
-- Module      : FileSynchronization
-- Description  : Expose a function which, for a given config
--               item, will read the corresponding file and determine the
--               current value as it would be identified by the pattern.
-- Copyright    : (c) Adrian Schurz, 2022
-- License      : MIT
-- Maintainer   : adrian.schurz@check24.com
-- Stability    : experimental
...
```

Zu diesem Zweck wurde über das Betriebssystem T_EX-Live [1] installiert und ein Plugin für die IDE [20]. Die Buildumgebung wurde so konfiguriert, dass bei jedem Speichern einer Datei das Programm kompiliert wird, sämtliche Softwaretests ausgeführt werden, sowohl die Moduldokumentation als auch der Graph der Modulabhängigkeiten neu generiert wird und Tools zur Formatierung und statischen Codeanalyse ausgeführt werden.

6.2 Entwicklung

Es wurde ein Stack[22]-Projekt angelegt und die resultierenden Verzeichnisse und Dateien wurden zur Versionskontrolle in einem Git[10]-Repository organisiert. Anschließend wurde das Repository auf dem unternehmenseigenen Bitbucket-Server [5] bereitgestellt.

6.3 Dokumentation

Das Projekt wurde sowohl auf Quellcodeebene zum Zwecke der Weiterentwicklung als auch für Nutzer bzw. Kunden dokumentiert (siehe Kapitel 9 - Kundendokumentation).

Code- und Moduldokumentation Jedes Modul und jede Deklaration im Code ist mit Codekommentaren versehen wie in den Abbildungen 3 und 4 darstellt. Anhand dieser Informationen wird mittels Haddock [15], während des Buildprozesses, eine navigierbare und übersichtliche Moduldokumentation im HTML-Format erstellt, die sowohl die Module des Programms selbst als auch die der verwendeten Softwarebibliotheken beschreibt und im Projektverzeichnis unter

```
/doc/generated
```

abgelegt ist.

Abbildung 4: Beispiel eines Deklarations-Codekommentars

```
-- | A function to change the cursor position to point at the
--   next item.
selectNextItem :: AppState -> AppState
...
```

Projektdokumentation Die Dokumentation der Projektarbeit selbst erfolgt mittels \LaTeX [19] und wird im PDF-Format exportiert. Alle Quelldateien für dieses Dokument sind ebenfalls im Projektverzeichnis unter

```
/doc/azubi-project/projektdokumentation
```

auffindbar.

6.4 Qualitätssicherung

Zur Qualitätssicherung dienen hauptsächlich Unittests, inklusive Property-Based-Testing, und händische Tests.

Unittests Ein Beispiel für einen Unittest ist in Abb. 5 gegeben. Damit wird pro Test die Ausgabe einer Funktion für einen speziellen Eingabewert mit einem korrekten Ausgabewert verglichen. Diese Art des Testens einzelner Codesegmente ist sinnvoll, aber aufgrund der beschränkten menschlichen Kreativität nicht besonders gut geeignet um Grenzfälle mit besonders großen, besonders absurden oder anderweitig unerwarteten Eingabewerten ausfindig zu machen.

Property-Based-Testing Um die obengenannte Schwäche von reinen Unittests auszugleichen wurden zusätzlich sogenannte Propertytests verfasst, welche mit einer großen Anzahl von zufällig generierten Eingabewerten bestimmte Eigenschaften der Ausgabewerte prüfen. In Abb. 6 ist ein Test illustriert, der sicherstellt, dass die Zielfunktion, unabhängig von sowohl ihrem Inhalt als auch dem konfigurierten Zielmuster, unverändert bleibt wann immer der aktuelle Wert und der neue Zielwert identisch sind. Zu jeder Ausführung der Testsuite werden dafür eine große Anzahl zufälliger Dateiinhalte und Zielmuster generiert und die Funktion damit geprüft.

Akzeptanztests Bestimmte Klassen von Softwarefehlern lassen sich allein mit Unittests nicht zuverlässig ausschließen. Für die umfänglichste Überprüfung der Software sind Tests, welche die Ausführung des kompilierten Programms, inklusiver simulierter Nutzerinteraktion, prüfen, von hohem Wert. Es wurde versucht mit Mitteln des Unittesting in Bash [4], virtualisierten Betriebssystemen [24] [7] bzw.

Abbildung 5: Beispiel eines Unittests, aus Platzgründen mit ... eingekürzt und inklusive der Ausgabe bei Ausführung (unterhalb des Pfeils)

```
describe "changing the element at a certain index inside of a list" $ do
  it "given a valid index ... apply it at the appropriate index" $
    let someList = [1, 2, 3]
      someFunction = (*) 2
      validIndex = 1
    in changeNthElement validIndex ... someList `shouldBe` [1, 4, 3]

↓

changing the element at a certain index inside of a list
  given a valid index ... apply it at the appropriate index [✓]
```

Abbildung 6: Beispiel für Property-Based-Testing, aus Platzgründen mit ... eingekürzt und inklusive der Ausgabe bei Ausführung einer großen Anzahl automatisch generierter Tests (unterhalb des Pfeils)

```
describe "modifying a string according to a search pattern ..." $ do
  prop "... old and new values are identical ... content unchanged" $
    \tva pat cont -> modify tva tva pat cont == cont

↓

modifying a string according to a search pattern ...
  given that the old and new values are ... content unchanged [✓]
  +++ OK, passed 1000 tests.
```

Terminalemulatoren und Tools zur Emulation von Tastatureingaben [30] [32] solche Test zu realisieren. Das Ergebnis ist in Kapitel 7.4 - Testabdeckung dokumentiert. Anstelle von Akzeptanztests, welche nicht automatisiert durchgeführt werden konnten, wurden regelmäßig händische Tests des kompilierten Programms auf dem Entwicklungsrechner durchgeführt. Ab einem gewissen Zeitpunkt während der Entwicklung befand sich das Programm bereits in einem nutzbaren Zustand. Es wurde vom Entwickler selbst von da an produktiv eingesetzt was, ähnlich dem Konzept des Betatestings, eine praktisch wertvolle, wenn auch wenig rigorose, Testabdeckung ermöglicht hat.

7 Ergebnisdiskussion

7.1 Funktionalität

Die Hauptfunktionalität des Programms wurde erfolgreich umgesetzt und es wird von mir selbst bereits genutzt. Das Programm arbeitet seither wie erwartet, ist konfigurierbar und informiert mit lesbaren Nachrichten im Falle eines Fehlers. Ein weiteres Feature wäre jedoch wünschenswert, die Fähigkeit des Programms mit schreibgeschützten Zieldateien umgehen zu können und in diesem Fall eine sudo-Passwortabfrage auszulösen. Letzteres ist nicht Teil der ursprünglichen Anforderungen und wird daher erst zukünftig umgesetzt.

7.2 Domain Driven Design

Eines der Nebenziele war es, die Abhängigkeiten der einzelnen Softwaremodule untereinander einem speziellen Schema folgen zu lassen (siehe Paragraph Softwaredesign in Kapitel 5.3). Die ist mit einer Ausnahme geglückt. Im Zielschema im Anhang, Abb. 9, sind Module der Applikationsebene ausschließlich abhängig von Modulen des User-Interface. Das verwendete GUI-Framework, Brick, führt durch sein Interface allerdings zu einem Muster bei dem diese Abhängigkeit umgekehrt ist. Letzteres wird deutlich beim Vergleich des Schemas mit den tatsächlichen Modulabhängigkeiten (siehe Abb. 10 im Anhang). Dieser Umstand stellt für den Moment eine vernachlässigbare Designschwäche dar die ohne weitere Konsequenz ist. Aus diesem Grund erhielt die Aufgabe dies zu beheben eine niedrige Priorität und steht vorerst noch aus. Alle sonstigen Module halten das angestrebte Schema ein.

7.3 Moduldokumentation

Das Ziel die Moduldokumentation ständig, während des Buildprozesses, aktuell zu halten ist teilweise geglückt. Im Projektverzeichnis liegt die ausführliche Moduldokumentation einschließlich jener der eingebundenen Softwarebibliotheken im HTML-Format vor. Auszüge davon sind in den Abbildungen 7 und 8 dargestellt. Allerdings gibt es seit wenigen Tagen bei neueren Versionen des Buildsystems Stack [22] und des Generierungstools Haddock [15] eine Inkompatibilität. Bis diese in diesen Projek-

ten behoben und veröffentlicht ist, ist die in diesem Projekt hinterlegte Moduldokumentation nicht aktuell. Es ist zu erwarten, dass dieses Problem in naher Zukunft seitens der Entwickler der beiden Tools behoben wird.

7.4 Testabdeckung

Die Abdeckung der Funktionalität des Programms auf Unittest-Ebene ist, besonders dank der Verwendung von Property-Based-Testing [17] zufriedenstellend. Einen grobe Überblick über die Unittests der einzelnen Module gibt eine Beispielausgabe der Ausführung der Testsuite in Abb. 5 im Anhang.

Eines der angestrebten Ziele war jedoch neben ausführlichen Unittests ebenso ausführliche Akzeptanztests bereitzustellen. Dies ist trotz erheblichem Aufwand gescheitert. Das Programm ist eine Konsolenanwendung, welche auf Tasteneingaben reagiert. Das bedeutet, dass automatisierte Tests in einer kontrollierten Umgebung Terminal-Emulatoren starten und ihnen Tasteneingaben simulieren müssen um das Programm zu testen. Je nach der verwendeten grafischen Benutzeroberfläche des Betriebssystems unterscheiden sich die Ansätze dies zu erreichen jedoch stark. Es existieren mehr oder weniger gut gepflegte Open-Source-Softwaretools für die Teilaufgabe der Eingabeemulation. Verschiedene Kombinationen dieser Tools (xdotool [30] vs ydotool [32]) wurden mit verschiedenen Displayservern (xorg [28] vs. wayland [26]), Betriebssystemen (Ubuntu [23] vs Arch Linux [3]), und Virtualisierungslösungen (virtualbox [24] vs docker [7]) evaluiert. Keine der Varianten führte zum Erfolg.

7.5 Bekannte Fehler

Es sind zwei Bugs in Software-Abhängigkeiten des verwendeten GUI-Frameworks bekannt, welche mit geringer Häufigkeit das Rendering bzw. den Start des Programms beeinträchtigen. Diese Bugs sind dokumentiert (siehe [12] und [11]). Einer dieser Fehler führt dazu, dass der aktuelle Wert eines Eintrags weiß statt blau gerendert wird und ist schwer reproduzierbar (ix ca. pro 50-100 Programmstarts). Der andere Fehler führt zu einem Crash des Programms beim Start (ix ca. pro 30-50 Programmstarts). Beide Softwarefehler befinden sich in eingebundenen Open-Source-Softwarebibliotheken. Bugfixes dieser Projekte werden, sobald sie verfügbar werden, für dieses Projekt übernommen.

8 Anhang

Abbildung 7: Generierte Moduldokumentation, Hauptseite

terminal-config-manager-0.1.0.0

Quick Jump · Instances · Contents · Index

terminal-config-manager-0.1.0.0

Please see the README on GitHub at <https://github.com/githubuser/terminal-config-manager#readme>

Modules

- ▼ Application
 - [Application.App](#) Build and run a Brick app after loading the config file.
- ▼ Domain
 - [Domain.State](#) Type definitions concerning the application state.
 - [Domain.StateTransition](#) Expose functions to change the application state. This covers item and value selection.
- ▼ Infrastructure
 - [Infrastructure.Config](#) Expose a function which parses the contents of the config file and returns a list of items specified in that file.
 - [Infrastructure.FileModification](#) Expose a type representing target file content and a function to modify the content of a file.
 - [Infrastructure.Util](#) Expose a range of helper functions.
- ▼ UserInterface
 - [UserInterface.Input](#) Handle the keyboard user input.
 - [UserInterface.Render](#) Expose a function to draw an application state to the screen and two different styling, one for the description text and one for the value

Produced by [Haddock](#) version 2.26.0

Abbildung 8: Generierte Moduldokumentation, Domain.State

Domain.State

Documentation

data AppState# Source

The state of the application. It will guarantee that the list of items contains at least one element and keep track of a cursor position.

Constructors

MkAppState (NonEmptyCursor ConfigItem)

Instances

▷ Show AppState# Source

▷ Eq AppState# Source

type NextAppState = EventM ResourceName (Next AppState)# Source

When reacting to an event the result needs to be of type EventM. Though not entirely accurate the synonym here is trying to communicate that, in essence, a new state of the application is created.

data ResourceName# Source

An dummy type used only as a parameter of EventM

Instances

▷ Show ResourceName# Source

▷ Eq ResourceName# Source

▷ Ord ResourceName# Source

Synopsis

13

Abbildung 9: Beispielschema für Modulabhängigkeiten beim Domain-Driven-Design [9]

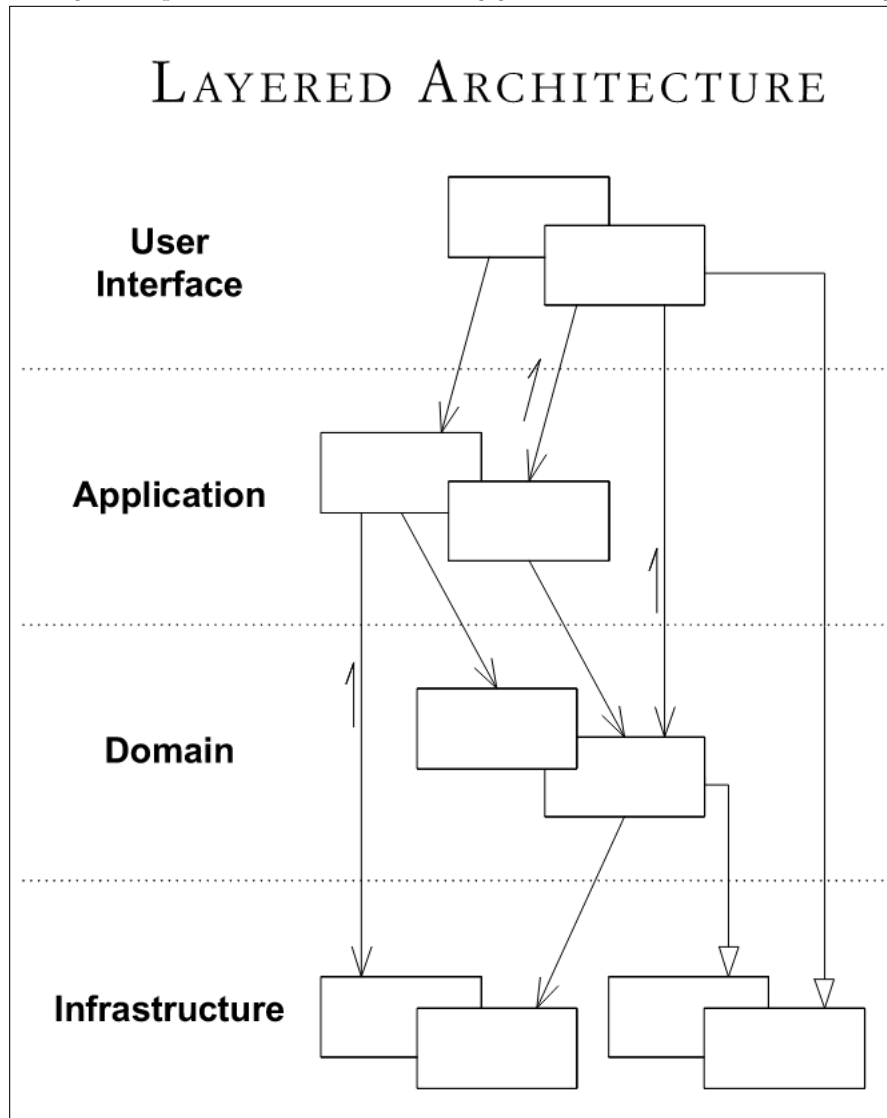


Abbildung 10: Graph der Modulabhängigkeiten

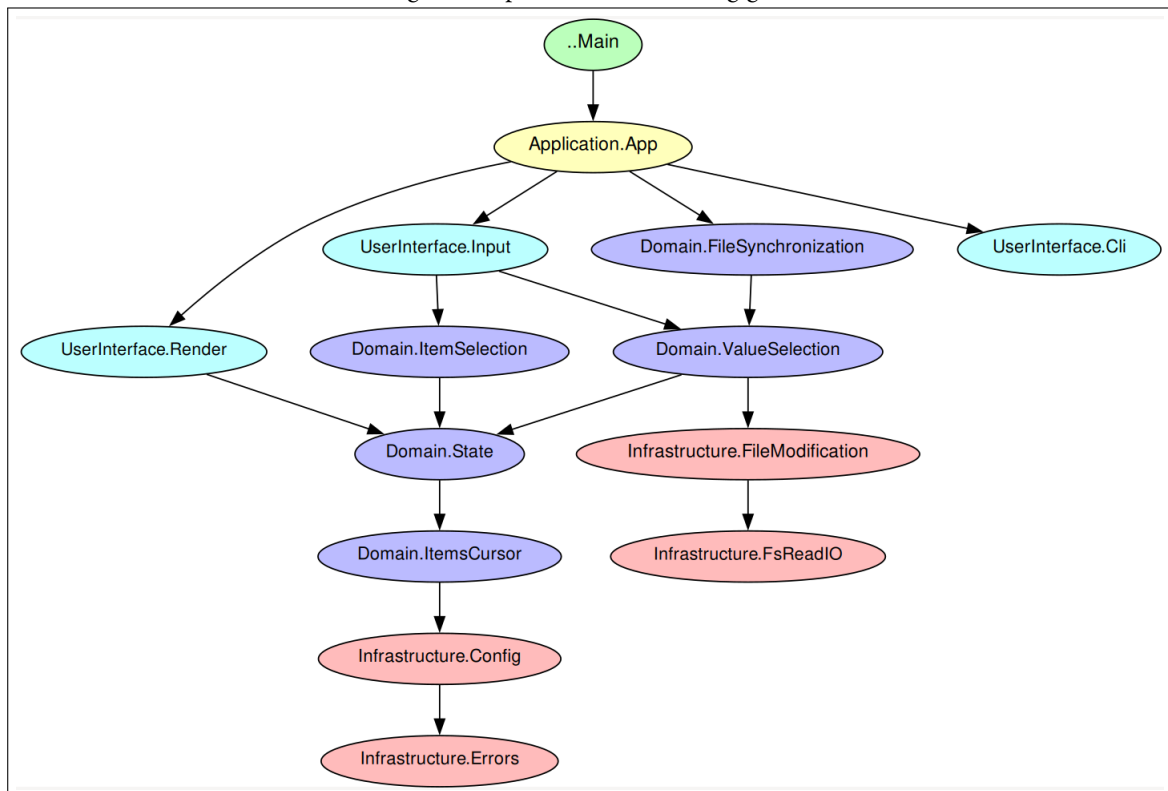


Abbildung 11: Ausgabe bei Ausführung der Testsuite

```
Unit.Domain.FileSynchronization
Finding the current value inside of some file content
  given completely empty input, should result in Nothing [✓]
  given a pattern and nothing else, should result in Nothing [✓]
  given file content and nothing else, should result in Nothing [✓]
  given a value marker and nothing else, should result in Nothing [✓]
  given a pattern without text in front or after the value marker, should result in the entire content [✓]
  given a pattern which matches at the start of the content, should result in the rest of the content without the pattern before the value marker [✓]
  given a pattern which matches a few characters into the content, should result in the rest of the content without the pattern before the value marker [✓]
  given a pattern with nothing after the value marker, should result in a match only up to and not including the next newline [✓]
  given a pattern with nothing before the value marker, should result in a backward match only up to and not including the nearest newline [✓]
Remove a substring from a given string and everything before it
  given completely empty input, should result in the empty string [✓]
  given a prefix, should remove it [✓]
  given a substring which is not a prefix, should remove it and everything before it [✓]
Unit.Domain.ItemsCursor
  Creating a list of items
    given an empty list should result in Nothing [✓]
    given a nonempty list should succeed and set the default cursor position to zero [✓]
  Moving the cursor
    upwards should increment the cursor position [✓]
    upwards and then downwards should leave the itemCursor unchanged [✓]
    should stop at the upmost item [✓]
    should stop at the bottommost item [✓]
  Changing the item at the cursor position
    given multiple items with the cursor position somewhere in the middle should modify the item at the correct position [✓]
Unit.Domain.ValueSelection
  selecting the element after a given value
    given an empty list should default to the target value [✓]
    given a list containing only the target value should return it [✓]
    given a first element equal to the target value should return the second element [✓]
    given the target value somewhere in the middle should return the successor [✓]
    given the target value as the last element should wrap around to the first one [✓]
    with the target value missing in the list should return the first element of the list [✓]
  selecting the element before a given value
    given an empty list should default to the target value [✓]
    given a list containing only the target value should return it [✓]
    given a list containing the target somewhere in the middle, should return the element before it [✓]
    given the target value as the first element should wrap around to the last one [✓]
  changing the element at a certain index inside of a list
    given index 0, an empty list and id as a function should do nothing [✓]
    given a negative index, should do nothing [✓]
    given a valid index and a function should apply it at the appropriate index [✓]
    given an index exceeding the length of the list should do nothing [✓]
  modifying a string according to a search pattern and a new value
    given empty content should leave the content unchanged [✓]
    +++ OK, passed 1000 tests.
    given an empty pattern should leave the content unchanged [✓]
    +++ OK, passed 1000 tests.
    given that the old and new values are identical should leave the content unchanged [✓]
    +++ OK, passed 1000 tests.
    given a pattern consisting only of the value marker should substitute all occurrences [✓]
    given a prefix to the value marker should only modify the occurrence having said prefix [✓]
    given a suffix to the value marker should only modify the occurrence having said suffix [✓]
    given all empty inputs should result in empty content [✓]
    given all parameters being empty besides the first should not crash [✓]

Finished in 0.0715 seconds
45 examples, 0 failures
```

Abbildung 12: Verzeichnisstruktur des Projekts

```
\src
\test
\doc
\distribution
\bin
\tool
package.yaml
README.md
```

9 Kundendokumentation

9.1 Beschreibung

Terminal-Config-Manager ist ein Linux-Programm mit welchem Textpassagen innerhalb mehrerer Dateien schnell zwischen einer Reihe von vorkonfigurierten Textpassagen umgeschaltet werden können. Der Hauptanwendungsfall ist die effiziente Manipulation von Konfigurationsdateien von Softwareanwendungen die häufig angepasst werden müssen.

9.2 Installation

Es wurden vorkonfigurierte Pakete für sowohl ArchLinux[3]-basierte als auch Debian[6]-basierte Betriebssysteme bereitgestellt. Alternativ kann das Programm auch manuell installiert werden.

Arch-Linux [3], via PKGBUILD Datei und pacman [21]

Im Projektverzeichnis unter

```
/distribution/arch/PKGBUILD
```

befindet sich eine Spezifikationsdatei anhand derer das Softwarepaket erstellt und anschließend installiert werden kann:

```
cd distribution/arch
makepkg
pacman -U terminal-config-manager-1.0.0-1-x86_64.pkg.tar.zst
```

Die Deinstallation erfolgt mittels

```
pacman -R terminal-config-manager
```

Debian, via .deb Datei und dpkg[8] bzw. apt[2]

Im Projektverzeichnis unter

```
/distribution/debian/terminal-config-manager.deb
```

befindet sich ein Softwarepaket, das mittels dpkg oder apt direkt installiert werden kann.

```
cd distribution/debian
dpkg --install ./terminal-config-manager.deb
# apt install ./terminal-config-manager.deb
```

Die Deinstallation erfolgt mittels

```
dpkg --remove terminal-config-manager  
# apt remove terminal-config-manager
```

Alternative, ohne Paketmanager

Wenn das Programm nicht vom systemeigenen Paketmanager verwaltet werden soll, dann kann es manuell kompiliert und in einem passenden Verzeichnis abgelegt werden.

Voraussetzung hierfür ist, dass das Programm `stack` auf dem System installiert ist.

Im Projektverzeichnis wird mit

```
stack build --test --copy-bins
```

das Programm kompiliert, die Testsuite ausgeführt und die ausführbare Datei im Projektverzeichnis unter

```
bin/terminal-config-manager
```

abgelegt. Anschließend kann das Programm in ein Verzeichnis kopiert werden, das in die Systempfadliste eingetragen ist, beispielsweise

```
cp bin/terminal-config-manager ~/.local/bin
```

Die Deinstallation erfolgt mittels

```
rm ~/.local/bin/terminal-config-manager  
rm <Konfigurationsdateipfad>
```

9.3 Konfiguration

Die Zieldateien und -textpassagen müssen vor Ausführung des Programms über eine Datei im YAML-Format [31] konfiguriert werden.

Verzeichnis Das Programm erwartet, dass sich eine solche Datei in einem der folgenden Verzeichnisse befindet. Die Reihenfolge entspricht der absteigenden Priorität beim Vorhandensein mehrerer Konfigurationsdateien:

1. `./config.yaml`

Abbildung 13: Beispielaufbau der Konfigurationsdatei

```
config_lines_to_manage:
- title: Beispieltitel 1
  path: /home/alice/zieldatei.conf
  pattern: "'statspush_enabled' => {{value}},"
  targetValue: "true"
  possibleValues:
    - "true"
    - "false"

- title: Beispieltitel 2
  path: /home/alice/verzeichnis/weitere-zieldatei.txt
  pattern: "SOFTWARE_ENV={{value}}"
  targetValue: production
  possibleValues:
    - testing
    - staging
    - production
    - local

- ...
```

2. `${HOME}/.config/terminal-config-manager/config.yaml` (**empfohlen**)
3. `${HOME}/.terminal-config-manager.yaml`

Der Dateipfad 1 bezeichnet den Ort der ausführbaren Datei selbst und sollte nur zu Debugging- oder Entwicklungszecken genutzt werden. Die Pfade 2 und 3 sind gängige Ablageorte für nutzerspezifische Konfigurationsdateien unter Linux und sind für die normale Nutzung geeignet.

Aufbau In Abb. 13 ist der Aufbau der Konfigurationsdatei illustriert.

- `config_lines_to_manage:`

Dies ist das äußere Element der Konfigurationsdatei und **muss** vorhanden sein. Innerhalb dessen befindet sich eine Liste von Einträgen. Jeder Eintrag gehört zu genau einer Textpassage, die mithilfe des Programms gezielt verändert werden soll.

Jeder Eintrag hat folgende Unterelemente:

- `title: Beispieltitel 1`

Dies ist ein Titel der frei gewählt werden kann und vom Programm angezeigt wird. Idealerweise wird dafür eine sehr kurze Beschreibung des von der Ziel-Textpassage gesteuerten Anwendungsverhaltens benutzt.

- `path: /home/alice/zieldatei.conf`

Hier wird der absolute Dateisystempfad der Zieldatei angegeben innerhalb derer Textpassagen verändert werden sollen.

- `pattern: "'statspush_enabled' => {{value}},"`

Dieses Feld enthält ein Zielmuster. Mithilfe dieses Musters das einen Platzhalter enthält wird vom Programm der genaue Ort der Ziel-Textpassage identifiziert. Das Zielmuster sollte sowohl den Platzhalter als auch an die Ziel-Textpassage angrenzenden Text enthalten. **Wenn die Kombination aus Platzhalter und angrenzendem Text die Ziel-Textpassage nicht eindeutig eingrenzt, dann wird vom Programm ausschließlich die erste übereinstimmende Textpassage modifiziert.**

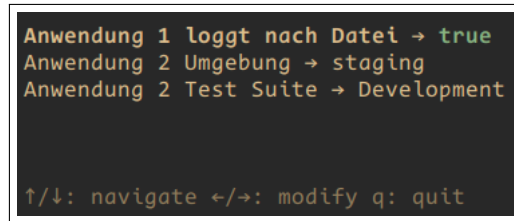
- `targetValue: "true"`

Der aktuelle Wert der Ziel-Textpassage. **Da das Programm in neueren Versionen beim Start den aktuellen Wert selbstständig ausliest, wird dieser Konfigurationswert in einer der folgenden Versionen entfernt werden. Aktuell muss er jedoch weiterhin in der Konfigurationsdatei angegeben werden.**

- `possibleValues:`
 - `"true"`
 - `"false"`

Dies ist die Liste der Werte die mithilfe des Programms ausgewählt und anstelle der Ziel-Textpassage in die Zieldatei geschrieben werden können. **Wenn sich der aktuelle Wert nicht in dieser Liste befindet, dann kann mit dem Programm auch nicht auf den ursprünglichen Wert zurückgewechselt werden.**

Abbildung 14: Ansicht nach Programmstart



```
Anwendung 1 loggt nach Datei → true
Anwendung 2 Umgebung → staging
Anwendung 2 Test Suite → Development

↑/↓: navigate ←/→: modify q: quit
```

9.4 Benutzung

Start Das Programm wird nach erfolgreicher Installation mit dem Befehl

```
terminal-config-manager
```

von der Kommandozeile aus gestartet. Für jeden Eintrag in der Konfigurationsdatei zeigt das Programm eine Zeile an.

Ansicht Abb. 14 zeigt eine typische Ansicht direkt nach dem Start des Programms. Die ersten drei Zeilen repräsentieren jeweils einen Eintrag in der Konfigurationsdatei und somit eine Ziel-Textpassage mit ihrem aktuellen Wert. Jede dieser Zeilen besteht aus dem in der Konfigurationsdatei vergebenen Titel des Eintrags, einem Pfeil und dem aktuellen Wert der Ziel-Textpassage. Die **aktuell ausgewählte Zeile** ist fett gedruckt während der **aktuelle Wert** der ausgewählten Zeile blau dargestellt wird. Die genaue Darstellung ist dabei vom verwendeten Terminal-Emulator und dessen Einstellungen bezüglich Schriftart und Farbwerten abhängig.

Die ausgegraute Zeile am unteren Rand zeigt zu jeder Zeit in Kurzform die verfügbaren Kommandos und die davon ausgelösten Aktionen.

Aktionen Die Hauptfunktionen werden über die vier Pfeiltasten gesteuert. Die Pfeiltasten hoch bzw. runter bewegen die Zeilenmarkierung nach oben bzw. unten. Die Pfeiltasten links bzw. rechts schalten den zur markierten Zeile gehörigen Wert weiter zum nächsten Wert aus der konfigurierten Liste der möglichen Werte (siehe Kapitel 9.3 - Konfiguration). Beim Umschalten eines Werts wird die zugehörige Zielfeile entsprechend modifiziert.

Mit einem Tastendruck auf q kann das Programm jederzeit beendet werden und zur Kommandozeile zurückgekehrt werden.

9.5 Problembehandlung

Fehlende Konfigurationsdatei Wenn beim Programmstart der Fehler

```
Error: No config file found at any of the search paths: ...
```

auftritt, dann bedeutet das, dass bei der Suche nach Konfigurationsdateien an keinem der angegebenen Pfade eine Datei gefunden wurde.

Lösung Es wird wie in Kapitel 9.3 (Konfiguration) beschrieben eine Konfigurationsdatei an einem der validen Dateipfade angelegt. Im Dateisystempfad unter

```
/usr/share/terminal-config-manager/config.yaml
```

befindet sich eine Beispielskonfigurationsdatei, welche als Vorlage genutzt werden kann:

```
mkdir ~/.config/terminal-config-manager  
cp /usr/share/terminal-config-manager/config.yaml \\  
~/.config/terminal-config-manager
```

Falsches Konfigurationsdateiformat Wenn beim Programmstart ein Fehler ähnlich

```
An error occurred while parsing the configuration file.  
The details are: ...
```

auftritt, dann bedeutet das, dass die erste vom Programm gefundene Konfigurationsdatei entweder nicht dem YAML-Format [31] entspricht und/oder fehlende Elemente aufweist.

Lösung Die Fehlermeldung wird weitere Detailinformationen enthalten wie beispielsweise:

```
The top level of the config file  
should be an object named 'config_lines_to_manage'
```

anhand derer sich das Problem identifizieren lässt. Im Zweifelsfall muss dem Kapitel 9.3 (Konfiguration) bzw. der Problembehandlung für fehlende Konfigurationsdateien in Kapitel 9.5 folgend eine valide Konfigurationsdatei per Hand angelegt werden.

Fehlende Dateizugriffsrechte Wenn bei der Auswahl eines neuen Werts der Fehler

```
terminal-config-manager: <Zielfeldpfad>: withFile:  
permission denied
```

auftritt, dann bedeutet das, dass dem aktuellen Linux-Nutzer die nötigen Zugriffsrechte fehlen um die Zielfeld zu modifizieren.

Lösung Es muss sichergestellt werden, dass alle in der Konfigurationsdatei definierten Zielfeldern vom aktuellen Nutzer modifiziert werden können. Im häufigsten Fall ist der aktuelle Nutzer nicht als owner der Datei eingetragen:

- Wenn dies angebracht ist, dann kann der Nutzer der Datei geändert werden:

```
chown <Nutzer> <Zielfeldpfad>
```

- Wenn dies angebracht ist, dann können die Schreibrechte der Zielfeld angepasst werden:

```
chmod o+w <Zielfeldpfad>
```

Wenn keine der beiden oben genannten Optionen anwendbar ist, dann ist diese Zielfeld nicht für die Modifizierung durch das Programm geeignet.

Literatur

- [1] *T_EX-Live Project Homepage*. URL: <https://www.tug.org/texlive>.
- [2] *Apt Paketmanager Project Homepage*. URL: <https://wiki.debian.org/AptCLI>.
- [3] *Arch Linux Project Homepage*. URL: <https://archlinux.org>.
- [4] *Bats-core: Bash Automated Testing System*. URL: <https://github.com/bats-core/bats-core>.
- [5] *Bitbucket Project Homepage*. URL: <https://bitbucket.org/product>.
- [6] *Debian Project Homepage*. URL: <https://www.debian.org>.
- [7] *Docker Project Homepage*. URL: <https://www.docker.com>.
- [8] *dpkg Paketmanager Manual*. URL: <https://man7.org/linux/man-pages/man1/dpkg.1.html>.
- [9] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. 1. Aufl. Addison-Wesley, 2003. Kap. 4 - Isolating the Domain, S. 68.
- [10] *Git Project Homepage*. URL: <https://git-scm.com>.
- [11] *Github Issue eines Bugs der selten zu einem Rendering-Problem führt*. URL: <https://github.com/judah/terminfo/issues/47>.
- [12] *Github Issue eines Bugs der zum Crash beim Programmstart führen kann*. URL: <https://github.com/jtdaugherty/vty/issues>.
- [13] *Glasgow Haskell Compiler Homepage*. URL: <https://www.haskell.org/ghc>.
- [14] *Graphmod Project Homepage*. URL: <https://github.com/yav/graphmod>.
- [15] *Haddock Project Homepage*. URL: <https://haskell-haddock.readthedocs.io/en/latest>.
- [16] *Haskell Project Homepage*. URL: <https://www.haskell.org>.
- [17] Fred Herbert. *The Pragmatic Programmers: Property-Based Testing with PropEr, Erlang, and Elixir*. 1. Aufl. Pragmatic Bookshelf, 2019.
- [18] *Jinja2 Project Homepage*. URL: <https://jinja.palletsprojects.com/en/2.11.x>.
- [19] *Latex Project Homepage*. URL: <https://www.latex-project.org>.
- [20] *Latex-Workshop Project Homepage*. URL: <https://github.com/James-Yu/LaTeX-Workshop>.
- [21] *Pacman Project Homepage*. URL: <https://archlinux.org/pacman>.

- [22] *Stack Project Homepage*. URL: <https://docs.haskellstack.org/en/stable>.
- [23] *Ubuntu Project Homepage*. URL: <https://ubuntu.com>.
- [24] *VirtualBox Project Homepage*. URL: <https://www.virtualbox.org>.
- [25] *Visual Studio Code Project Homepage*. URL: <https://code.visualstudio.com>.
- [26] *Wayland Project Homepage*. URL: <https://wayland.freedesktop.org>.
- [27] *Wikipediaartikel, Terminalemulation*. URL: <https://de.wikipedia.org/wiki/Terminalemulation>.
- [28] *X Server Project Homepage*. URL: <https://www.x.org>.
- [29] *XDot Project Homepage*. URL: <https://github.com/jrfonseca/xdot.py>.
- [30] *xdotool Project Homepage*. URL: <https://github.com/jordansissel/xdotool>.
- [31] *YAML Project Homepage*. URL: <https://yaml.org>.
- [32] *ydotool Project Homepage*. URL: <https://github.com/ReimuNotMoe/ydotool>.

Glossar

Betatesting "Der Begriff Betatest bezeichnet den Softwaretest eines Software-Produktes im Entwicklungsstadium einer Beta-Version, der unter möglichst realen Anwendungssituationen von späteren Benutzern („Nachfrager“) durchgeführt wird." - abgewandelter Auszug aus **[beta-testing]**

I, IO

GUI Abkürzung für den Begriff graphical user interface. Zu deutsch: grafische Benutzeroberfläche.

IO, II

IDE Abkürzung für den englischen Begriff integrated development environment - integrierte Entwicklungsumgebung

I, 2, 6, 7

PDF Abkürzung für den englischen Begriff portable file format. Es bezeichnet ein weit verbreitetes Dokumentformat.

I, 8

Platzhalter Ein vordefinierter Text: das englische Wort `value` umgeben von doppelten geschweiften Klammern: `{{value}}` Das Format des Platzhalters ist an die Template-Engine Template-Engine Jinja2 [18] angelehnt. Er ist Teil des Zielmusters das zusätzlich Text vor und nach der Ziel-Textpassage enthält. Er markiert den Ort der Ziel-Textpassage relativ zum Zielmuster innerhalb der Zielfile. I, 20, 27

Property-Based-Testing Property-Based-Testing bezeichnet eine spezielle Strategie Softwaretests zu formulieren. Statt einer Reihe explizit ausformulierter Tests wird eine Regel formuliert, welche das zu testende Programm für automatisch generierte Eingabewerte einhalten muss. Die in großer Zahl automatisch generierten Testfälle decken mit höherer Wahrscheinlichkeit Grenzfälle durch z.B. besonders große oder abwegige Eingabewerte ab. I, 3, 8, 9, II

Rendering Im Kontext dieses Programms beschreibt der Begriff Rendering den Prozess der Wandlung eines Zustands des Programms in eine auf dem Bildschirm darstellbare Form, in diesem Fall Text. I, II

Terminalemulator "Ein Terminalemulator ist ein Computerprogramm, das die Funktion eines Computer-Terminals nachbildet. Sie wird genutzt, um textbasierte Programme innerhalb einer grafischen Benutzeroberfläche verwenden zu können." - abgewandelter Auszug aus [27] I, IO

Textpassage Ein Stück Text aus einer Textdatei. Dabei wird sich meist auf eine in der Konfigurationsdatei spezifizierte Zielfile des Programms bezogen. I, 17, 19–21, 26, 27

Zielmuster Ein Element innerhalb der Konfigurationsdatei. Es bezeichnet ein Stück Text das einen speziellen Platzhalter, den Text `{{value}}`, enthält. Es wird vom Programm dazu genutzt den genauen Ort der Ziel-Textpassage innerhalb der Zieldatei zu identifizieren. Das Zielmuster sollte

1, 8, 20, 26