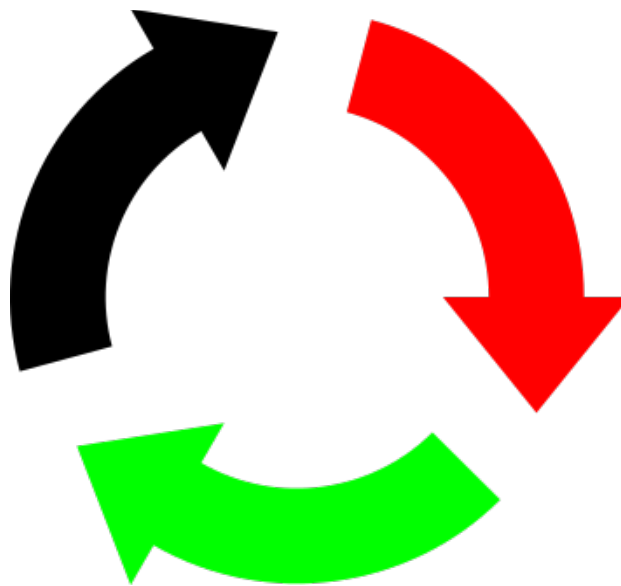


TDDT-Handbuch



Abschlussprojekt des Programmierpraktikums 2016 an der
Heinrich-Heine-Universität Düsseldorf

von Let's play ProPra

1. Inhaltsverzeichnis

1. Inhaltsverzeichnis
2. Was ist TDD und was macht dieses Programm?
3. Menü-Leiste
4. Phasen
 - Test
 - Coding
 - Refactor
5. Features
6. Sammlung von Fehlerausgaben

2. Was ist TDD und was macht dieses Programm?

Test-driven development (TDD) ist eine Methode, um Programme zu entwickeln. Dabei schreibt man abwechselnd Tests und Codes in der folgenden Reihenfolge, bis das Programm fertig ist und das tut, was man möchte:

1. Zunächst schreibt man einen fehlschlagenden Test, der auf eine gewünschte Eigenschaft hin testet.
2. Ändere den Code so, dass der Test nicht mehr fehlschlägt.
3. „Verschönere“ den Code, indem bereits implementierte Dinge sinnvoll zusammengefasst werden.

Dieses Programm ist dazu da, dem Nutzer das TDD nahe zu bringen, indem verschiedene Aufgaben gestellt und mittels des obigen Schemas gelöst werden müssen.

Im Folgenden werden wir näher auf den Programmaufbau eingehen und uns nicht weiter mit TDD befassen, da dies ein Handbuch zum TDD-Trainer (TDDT) sein soll und keine Anleitung für TDD ;)

Wer mehr erfahren möchte, kann auf diversen Internetseiten und Büchern schmökern, z.B. unter https://de.wikipedia.org/wiki/Testgetriebene_Entwicklung

3. Menü-Leiste

Die Menü-Leiste beinhaltet 4 Haupt-Punkte:

- **Menü:** Von dort kann man eine neue Übung wählen oder das Programm beenden
- **Katalog:** Hier wechselt man zwischen verschiedenen Katalogen und Aufgaben. Es besteht die Möglichkeit, selbst erstellte Kataloge zu laden.
- **Tracking:** Dort kann man sich die Phasenanalyse anzeigen lassen.
- **Hilfe:** Hier kommt man zur Aufgabenstellung, falls man sich verrennt oder nicht klar ist, was man dort implementieren soll.

4. Phasen

Wie oben schon beschrieben, besteht das TDD im wesentlichen aus 3 Phasen: Dem *Testschreiben*, dem eigentlichen *Coden* und dem *Refactor*-Schritt.

Test

Am Anfang eines Zyklus steht immer ein fehlschlagender Test. Die rechte Seite des Programms ist die Seite, die zu Beginn aktiv ist und in die die Tests geschrieben werden müssen.

Code

Schlägt genau ein Test fehl, darf man zur Code-Sektion wechseln. Hier schreibt man solange Code, bis der in der vorigen Phase fehlschlagende Test keinen Fehler mehr produziert.

Refactor

Zur Refactor-Phase darf man wechseln, wenn der Code kompiliert und kein Test fehlschlägt. Hier kann man (auch bei aktiviertem Babysteps) in Ruhe seinen Code verbessern. Wenn der Code dann immernoch kompiliert werden kann und auch kein Test verletzt wurde, kann wieder zu Test gewechselt werden und dieser Zyklus-Durchlauf ist beendet.

5. Features

Tracking

Wie oben schon bei der Menü-Beschreibung erwähnt, gibt es eine Phasenanalyse: Sie zeigt dem Nutzer die Zeit, die er in den einzelnen Phasen verbracht hat, sowie die Änderungen, die gemacht wurden, an, um ihm die Möglichkeit zu geben, bestimmte Verhaltensmuster beim Codeschreiben zu erkennen und eventuell zu verbessern.

Babysteps

Es gibt so genannte Babystep-Aufgaben. Diese besitzen eine zeitbeschränkte Code- sowie Testphase, sodass man möglichst kleinschrittig vorgeht. Wird die Zeit nicht eingehalten, so wird in die vorige Phase zurückgewechselt und die bis dato gemachten Änderungen der nicht-bestandenen Phase gelöscht.

6. Sammlung von Fehlerausgaben

Im Folgenden werden einige Fehlermeldungen, die auftreten können erklärt.

- **„Unable to find config.txt: [...]“:** Es konnte die config.txt Datei nicht geladen werden. Sie ist essenziell für den Start des Programms. Sie liegt normalerweise im Ordner build/libs/config. Sollte dort keine solche Datei vorhanden sein, wäre ein Workaround eine neue Datei mit dem Namen config.txt anzulegen, welche folgenden Inhalt hat:

```
gewaehlterKatalog##letzteAufgabe# #
```

Beachte das Leerzeichen zwischen den letzten beiden Doppelkreuzen

- **„Unable to load resources: [...]“:** Dieser Fehler tritt auf, wenn die .fxml-Dateien nicht geladen werden konnten, welche für die Darstellung der Oberfläche notwendig sind. Diese liegen üblicher Weise im Ordner build/resources/main/fxml.
- **„Fehler außerhalb: [...]“:** Hier liegt ein Problem bei Ausführen der externen Datei vor, welche den Quellcode kompiliert, um eventuelle Konsolenausgaben abzufangen.
- **„Fehler beim Loggen: [...]“:** Deine Änderungen konnten nicht in die entsprechende Datei log.txt geschrieben werden. Diese sollte für jede Aufgabe im Ordner build/libs/aufgaben/AUFGABENNAME angelegt werden.
- **„Konnte Aufgabenkatalog nicht laden: [...]“:** Bei der Übermittlung des Pfades des von dir ausgesuchten Kataloges ist ein Fehler aufgetreten. Versuche über die Menüleiste den Katalog erneut zu laden.
- **„Fehler beim Quellcodeladen: [...]“:** Der letzte Stand, bei dem die gewählte Aufgabe verlassen/ gespeichert wurde, konnte nicht wiederhergestellt werden.
- **„Unable to save: [...]“:** Der gewählte Katalog bzw. die gewählte Aufgabe konnte beim beenden des Programms nicht in der config.txt gespeichert werden.

Alle oben aufgelisteten Fehler, sind solche, die das Programm aus die Kommandozeile schickt, von der aus es gestartet wurde. Es sind Fehler, die den korrekten Ablauf des Programms bzw. der darin enthaltenen Features (der Tracker zum Beispiel) erheblich behindern.

Weitere Fehler, die auftreten können und auf der Programm-internen Konsole angezeigt werden, sind weniger dramatisch und selbsterklärend. Meist liegen dann Fehler im geschriebenen Code bzw. Test vor.