Handbuch zum TDD-Trainer

1 Vorwort

Du kannst programmieren, aber dein Programmierstil lässt zu wünschen übrig? Du produzierst Code, von dem große Teile im Endeffekt ungebraucht bleiben? Du kreierst massenhaft Code, verlierst oft den Überblick über dein Programm, und hast es satt immer wahnsinnig viele Zeilen Code durchzusuchen, um einen Fehler zu finden? Du findest Unit-Tests prinzipiell gut, weißt aber nicht genau, wann und wie du sie sinnvoll nutzen kannst?
Dann bist du genau richtig! Der TDD-Trainer soll dir beim Verbessern deiner Skills helfen! Er wird dir das testgetriebene Entwickeln spielend leicht beibringen. Möchstest du mehr über testgetriebene Entwicklung erfahren, so ist der Text von Frank Westphal (`http://www.frankwestphal.de/ TestgetriebeneEntwicklung.html`) zu empfehlen.

2 Erste Schritte

2.1 Eine Übung auswählen

Um dich mit dem TDD-Trainer vertraut zu machen, raten wir dir, dass du eine Übung auswählst, die ohne Babysteps oder Tracking lösbar ist.

Öffne zunächst die Anwendung. Du wirst sehen: es erscheint ein Fenster, in dem du Kataloge und Aufgabenstellungen auswählen kannst. Drücke auf `[...]` um einen Katalog mit Übungen auszuwählen. Nachdem ein Katalog geladen ist, kannst du dir die Aufgaben, die in der Liste erscheinen, ansehen, indem du sie anklickst. Im Textfeld erscheint jeweils die Aufgabenstellung der ausgewählten Übung.
Hast du dich für eine Übung entschieden, so kann es direkt losgehen! Klicke auf `Start`.

2.2 Der TDD-Trainer

Nun sollte der TDD-Trainer erschienen sein. Oben links kannst du jederzeit die Aufgabenstellung beim Lösen der Übung nachlesen. Die Anzeigen mittig und oben rechts werden dir bei der Lösung der Übung sehr hilfreich sein. Dort werden (gewollte und ungewollte) Compilerfehler und Testergebnisse angezeigt. Mittig findest du deinen Arbeitsbereich. Im Test-Editor wirst du deine Tests, und im Code-Editor deine Lösung der Übung implementieren. Die Status-Leiste ganz unten führt dich mit Anweisungen durch den Trainer. Außerdem färbt sie sich grün, wenn alle Tests laufen, und rot, wenn Compilerfehler auftreten oder Tests nicht bestanden sind.
Wie du siehst, sind der Test-Editor und der Code-Editor bereits mit einem Codeskelett gefüllt. Dieses (besonders der Klassenname!) darf zu keiner Zeit geändert werden! Außerdem ist die Methodensignatur deines ersten Tests schon vorhanden. Den Namen dieser Methode darfst du natürlich der Aufgabe entsprechend sinnvoll verändern. Schreibe also nun deinen ersten Test. Du bist fertig? Kompiliere deinen Code, indem du auf `Run -> Compile and Run` klickst. Nun sollte ein Compilerfehler in der Anzeige auftauchen. Da der Code nicht kompiliert, also insbesondere gar nicht ausführbar ist, ist die Status-Leiste rot. Nun bist du bereit für die Implementierung des Codes, damit der Test besteht/funktioniert. Teile dies dem Trainer mit, indem du auf `Navigate`

`-> Edit Code` klickst. Folge den Anweisungen der Status-Leiste, und mache den Test funktionsfähig. Ob die aktuelle Implementierung funktioniert, kannst du dabei jederzeit mit `Compile and Run` überprüfen. In dieser Phase sollst du lediglich deinen Test zum laufen bringen, ohne auf Schönheit und Effizienz zu achten. Wenn du so weit bist, dass dein Test läuft, so kannst du deinen Code verbessern (Refactor). Hierbei kannst du alle unschönen und unsauberen Codefragmente überarbeiten. Bist du fertig mit verbessern? Dann wähle `Navigate -> End Refactor`.
(WICHTIG! Sollte `End Refactor` ausgegraut sein, so liegt das daran, dass dein Code nicht kompiliert oder nicht alle Tests erfüllt sind. Du darfst erst die Refactor-Phase verlassen, sobald mit `Compile and Run` dein Code auf vollständige Funktionsfähigkeit überprüft wird!) Nun startet der Prozess von neu. Schreibe einen weiteren Test, der nicht erfüllt ist. Drücke `Compile And Run` und dann `Edit Code` usw. Dabei darf aber immer nur genau ein Test nicht funktionsfähig sein (oder Compilerfehler haben), um mit `Edit Code` zum Code-Editor zu wechseln.
Solltest du in der Situation sein, dass du beim Schreiben der Lösung deines Tests im Code-Editor merkst, dass der Test nicht funktionieren kann oder einfach falsch ist, so ist das kein Problem. Du kannst vom Code Fenster immer mit `Back to Test` in den Test zurück wechseln (außer du bist in Refactor). Beachte aber, dass die Änderungen, die du im Code-Editor vorgenommen hast, verfallen!

2.3 Lösungen speichern und Übung beenden

Hast du deine Aufgabe vollständig gelöst oder solltest du die Übung vorzeitig verlassen wollen, so kannst du deine Lösungen als java-Dateien speichern. Dies klappt mit `Navigate -> Save`. Verlässt du eine Übung mit `Navigate -> Exit` ohne zu speichern, so verfällt deine Lösung. Keine Sorge! Bevor du zurück zum Hauptmenü kommst, wirst du daran erinnert, dass du deine Lösung abspeichern solltest. Führst du die Übung zu einem anderen Zeitpunkt fort, so wird deine abgespeicherte Lösung geladen. (WICHTIG: Achte darauf, dass die java-Dateien in der Zwischenzeit nicht verändert wurden!)

3 Erweiterungen

Wenn du dich schon etwas zurecht gefunden hast, kannst du nun auch die Übungen angehen, die die Erweiterungen Babysteps und Tracking unterstützen. Diese werden automatisch bei entsprechender Übung eingeschaltet.

3.1 Babysteps

Die Erweiterung Babysteps soll dir beibringen, kleinschrittig zu denken und zu programmieren. Bevor du den Fehler begehst und komplexe Lösungen implementierst, setzen wir dir ein Zeitlimit, in welchem du den entsprechenden Test zum Laufen bringen sollst. Du sollst den Test schließlich NUR zum Laufen bringen, und nicht direkt schön gestalten. Für letzteres wirst du alle Zeit der Welt haben, denn die Refactor-Phase ist nicht limitiert.

Wählst du also eine Übung aus, die Babysteps unterstützt, so musst du dir zuerst ein Zeitlimit setzen. Über `Babysteps Settings` kannst du auswählen, ob du 2, 3 oder 5 Minuten zum Lösen haben möchtest. Wählst du eine Zeitspanne aus, so fängt es auch schon an. Das Lösen der Übung verläuft dabei analog wie oben. Falls du es jedoch nicht schaffst, deinen Test oder den Code zum Test innerhalb des Limits zu schreiben, so verfällt dieser und du wechselst in die vorherige Phase.

3.2 Tracking

Die Erweiterung Tracking soll dir darlegen, wo deine Probleme liegen. Durch eine Übersicht beim Beenden der Übung kannst du sehen, wie lange du in welcher Phase verbracht hast, und in welcher Phase Fehler aufgetreten sind. Mit Phase ist dabei gemeint, in welchem Editor du dich befandest beziehungsweise ob du mit Refactor deinen Code verschönert hast. Idealerweise solltest du verhältnismäßig mehr Zeit in Refactor, als in den anderen Phasen verbracht haben.
Wählst du also eine Übung aus, die Tracking unterstützt, so wird dein Programmierverhalten analysiert. Oben rechts kannst du sehen, ob die Funktion ein- oder ausgeschaltet ist. Beim Lösen der Übung ändert sich für dich nichts. Erst wenn du die Übung beendest und auf `Navigate -> Exit` klickst, werden dir ein Diagramm und drei Listen angezeigt. Im Diagramm kannst du auslesen, wie lange du in welcher Phase insgesamt verweilt hast. Die Listen zeigen dir, in welcher Phase du entsprechende Fehler gemacht hast.

3.3 Dokumentation

Hauptmenü

`[...]` - Katalog auswählen

`Beenden` - Anwendung beenden

`Start` - Trainer starten

Trainer

`Navigate -> Back to Test` - Vom Code-Editor zum Test-Editor wechseln, Änderungen werden verworfen

`Navigate -> Edit Code` - Vom Test-Editor zum Code-Editor wechseln; nur möglich, wenn Compilerfehler oder genau ein Test fehlschlägt

`Navigate -> End Refactor` - Refactor Beenden und zum Test-Editor wechseln; nur möglich, wenn REFACTOR und kein Compilerfehler und alle Tests funktionsfähig

`Navigate -> Save` - geschriebene Tests und Lösung speichern

`Navigate -> Exit` - zurück zum Hauptmenü

`Run -> Compile and Run` - kompiliert Tests und Code und führt Tests aus

`Babysteps Settings -> ?  Minutes` - Babysteps Limit setzen; nur möglich, wenn Übung Babysteps kompatibel

Wir wünschen dir viel Spaß und großen Lernerfolg mit dem TDD-Trainer!

Silja Mohrmann, Julian Schacht und Kathrin Parchatka