Test-Driven-Development

AVM Workshop Poster

Ambs, Brandenburger, Jerke, Klimpel, Tögel
HTWG Konstanz, Fakultät Informatik

Contact Information:
Fakultät Informatik
HTWG Konstanz
Alfred-Wachtel-Str. 8 78462 Konstanz

GitHub: msi-se/avm-workshop-tdd

Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Zusammenfassung
Hier kommt der Abstract hin.

Grundlagen und Ziele

Ablauf

Bei Test-Driven-Development (TDD) wird die Entwicklung von Software durch einen Zyklus aus Testen und Implementieren gesteuert. Dabei werden die Tests optimalerweise vor der Implementierung geschrieben, in Einzelfällen auch parallel dazu. Der genaue Ablauf eines Zyklus unterscheidet sich zwischen verschiedenen Quellen leicht, im Kern ist er aber immer gleich. In diesem Poster werden die typischen drei Schritten („red, green, refactor“ z. B. nach dem Vorgehen von „Agile-Ikone“ Kent Beck (vgl. [**?**] S. x)) um einen weiteren verbreiteten Schritt erweitert, um möglichst viel abzudecken.

- **0. Schritt: „Think“**
  Zunächst sollten die Anforderungen an die gerade zu implementierende Funktionalität klar sein. Wenn dies nicht der Fall ist, muss es erneute Absprachen geben. Der/die Entwickler:in sollte sich hier aus Sichtweise des Nutzers der Softwarekomponente in die Funktionalität hineinversetzen. In diesem Schritt muss auch entschieden werden, welche Anforderungen genau in diesem Zyklus umgesetzt werden sollen und welche erst in einer zukünftigen Iteration in Betracht gezogen werden.
- **1. Schritt: „Red“**
  Im ersten richtigen Schritt wird ein Test geschrieben, der alle Anforderungen an die Funktionalität abdeckt. Hierzu können verschiedene Testarten verwendet werden, z. B. Unit-Tests, Integrationstests, etc. Wichtig: Der Test muss fehlschlagen, da die Funktionalität noch nicht implementiert ist („red“). So wird sichergegangen, dass auch wirklich neue Funktionalität getestet und implementiert wird. Oft lassen sich die Tests noch nicht einmal kompilieren, da beispielsweise die zu testende Methode noch gar nicht existiert.
- **2. Schritt: „Green“**
  Bei der anschließenden Implementierung wird nun darauf geachtet, dass der Test so schnell wie möglich erfolgreich durchläuft („green“). Es kommt hier nur auf die Funktionalität an, die Implementierung muss nicht „schön/sauber“ sein („quick and dirty“), sprich „so wenig Code wie möglich“.
- **3. Schritt: „Refactor“**
  Sind alle Tests erfolgreich, wird „aufgeräumt“. Der Code wird optimiert, Duplikate werden entfernt und die Lesbarkeit wird verbessert. Dabei werden die Tests immer wieder ausgeführt, um sicherzustellen, dass die Funktionalität bestehen bleibt.

(vgl. [**?**] S. x, [**?**] S. 153, [**?**])

Vorteile

Nachteile

Bezug zum Agilen Manifest

Best Practices

Kleine unabhänige Tests

Beim Erstellen von Tests sollte darauf geachtet werden, dass sie klein und unabhängig sind. Das bedeutet, dass ein Test nur eine spezifische Funktionalität testet und nicht von anderen Tests abhängig ist. Dadurch wird die Wartbarkeit der Tests erhöht und die Ursache von Fehlern ist leichter zu finden.

So wenig Code wie nötig

Es sollte nur so viel Code geschrieben werden, wie erforderlich ist, um den Test zu bestehen. Dabei sollte vermieden werden, Code zu implementieren der nicht direkt mit dem aktuellen Testfall zusammenhängt. Dies trägt dazu bei, die Codebasis schlanker und besser wartbar zu machen, da sich jeder geschriebene Code auf spezifische Anforderungen und Funktionalitäten konzentriert.

Positve und Negative Tests

Beim Schreiben von Tests ist es wichtig, sowohl positive als auch negative Szenarien zu testen. Das bedeutet, dass sowohl Fälle getestet werden sollten, in denen der Code korrekt funktionieren soll, als auch solche, in denen er scheitern soll. So können mögliche Probleme mit dem Code erkannt und sichergestellt werden, dass er robust und zuverlässig ist.

Testen Testen Testen

Tests sollten immer wieder ausgeführt werden. So kann sichergestellt werden, dass der Code auch nach Änderungen noch korrekt funktioniert. Darüber hinaus ermöglicht es, Fehler frühzeitig zu erkennen und eine zuverlässige Codebasis aufrechtzuerhalten.

Tools

Fazit