Software-Testing

Lektionen 25 - 28

Zentrum für berufliche Weiterbildung

Agenda

- Prüfung
- Schier unüberwindbare Probleme lösen
- Unit Testing – Best Practices
- Übungen

Prüfung

- Zeit: 45min
- Nach Abschluss der Prüfung Klassenzimmer verlassen und Pause machen.
- Fragen?

Teile und herrsche!

Wie geht man Aufgaben/Aufträge im Bereich der Softwareentwicklung an, bei denen man den Wald vor lauter Bäumen nicht mehr sieht?

Ideen?

Analyse / Design / Implementierung

Um (für euch unüberwindbare) Probleme zu lösen, müsst ihr eine Strategie entwickeln. Diese Vorgehensmethodik soll das Problem in viele kleine Probleme zerlegen, dabei ist ein strukturiertes und nachvollziehbares Vorgehen sehr wichtig!

Ganz nach dem Motto: «Teile und herrsche!»

Analyse 1/3

- Analyse der Aufgabenstellung (Use Case, User Story, Übung, Aufgabe, usw.)
- bei Softwarefehlern: Analyse was funktioniert nicht wie gewünscht.
- usw.

Analyse 2/3

- Bei diesem Punkt geht es darum, das Problem/Aufgabe zu verstehen und in seinen eigenen Worten nieder zu schreiben. Beim Lösen von sehr trivialen Aufgaben macht man dies oft nicht, was dazu führt, dass man wenn es etwas komplizierter wird, keine Strategie an der Hand hat um die Probleme zu zerlegen.

Analyse 3/3

- Früher in der Schule beim Lösen von Dreisatzaufgaben mussten Sie immer "gegeben" und "gesucht" aufschreiben. Das entspricht der Analyse.

Design 1/2

- Wie muss ich die Software konkret anpassen, dass ich die gewünschten Features abbilden kann.
- Was muss ich anpassen, dass die Software korrekt funktioniert.

Design 2/2

- Hier wird es konkret. Auf Basis der Analyse macht man sich Gedanken, welche Klassen man braucht, resp. welche Statements (z.B. IF, Switch, usw.) hinzugefügt oder angepasst werden müssen.

Achtung! Sie verändern bei diesem Schritt nichts! Wir dokumentieren es nur. z.B. als UML-Diagramm (kann eine Handskizze sein). Screenshot mit Pfeilen des Problems, o.ä.

Implementierung 1/1

- Die Änderungen werden durchgeführt.
- Die Software wird getestet, anhand des «Gesucht» (Testorakel) aus der Analyse.

Wie man sehen kann, ist die eigentliche Implementierung der kleinste Teil der Arbeit eines "guten" Programmierers. "Zuerst Denken, dann handeln" ist die Devise. Was auffallen wird, ist dass man der Letzte ist, der in die Tasten grifft, jedoch der erste bei dem die Software funktioniert. Man musst viel weniger "umprogrammieren" oder basteln...

Allgemein

WICHTIG!!!!

Bei diesem Vorgehen geht es nicht darum, einen Papiertiger zu produzieren, sondern nur die relevanten Informationen aufzunehmen die es braucht um das Problem zu lösen.

Vermeiden Sie Logik in Tests 1/2

Wenn sich etwas aus der folgenden Auflistung in Ihrem Unit Test befindet, dann enthält Ihr Test Logik, die nicht dort sein sollte:

- switch-, if- oder else-Anweisungen
- foreach-, for- oder while-Schleifen

Quelle: «The Art of Unit Testing, S. 197, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

Vermeiden Sie Logik in Tests 2/2

Ein Test, der Logik enthält, prüft meist mehr als eine Sache auf einmal, was nicht empfehlenswert ist, denn der Test ist dann weniger lesbar und eher brüchig. Aber die Testlogik fügt auch Komplexität hinzu, die einen versteckten Bug enthalten kann.

Verwenden sie auch keine Try…Catch Statements

Quelle: «The Art of Unit Testing, S. 197, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

Testen Sie nur einen Belang 1/2

Ein Belang («Concern») ist, […] ein einzelnes Endresultat einer Unit of Work: ein Rückgabewert, eine Änderung des Systemzustands oder der Aufruf eines Third-Party-Objekts. Wenn Ihr Unit Test beispielsweise Asserts auf mehr als ein einzelnes Objekt setzt, dann testet er vielleicht mehr als einen Belang.

Quelle: «The Art of Unit Testing, S. 199, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

Testen Sie nur einen Belang 2/2

Mehr als einen Belang zu testen, klingt nicht so schlimm, bis Sie beschließen, Ihren Test zu benennen, oder überlegen, was passiert, wenn die Asserts für das erste Objekt fehlschlagen.

Quelle: «The Art of Unit Testing, S. 199, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

Factory-Methode statt setUp 1/1

Verwenden Sie statt des setUp-Attributs Factory-Methoden um die SUT's zu initialisieren.

ACHTUNG!

Niemals Fakes über diesen Initialisierungsmechanismus einem SUT zuweisen. Die Tests werden so sehr schlecht lesbar!

Quelle: «The Art of Unit Testing, S. 206-210, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

Mehrfache Asserts 1/1

Wenn Asserts fehlschlagen, dann werfen sie Ausnahmen. ([…]) Sobald eine Assert-Anweisung eine Ausnahme wirft, wird keine andere Zeile der Testmethode mehr ausgeführt. Das bedeutet, dass die beiden anderen Assert-Anweisungen niemals ausgeführt werden, sobald das erste Assert […] fehlschlägt.

Quelle: «The Art of Unit Testing, S. 217, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

Magic Results 1/1

Sie müssen eine Methode testen, die einen Status-Code zurückgibt, ob diese erfolgreich war. Gibt diese «-100» zurück, konnte das File nicht gefunden werden.

Wenn Sie nun in Ihrem Assert «-100» schreiben merkt kein Mensch was dies bedeutet. Definieren Sie daher dazu eine Konstante mit dem Namen «COULD_NOT_READ_FILE» und weisen Sie dieser -100 zu. So kann ach Ihr Kollege gut verstehen was passiert wenn er den Code Reviewed!

Quelle: «The Art of Unit Testing, S. 217, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

Übung 1

Vorbereitung

- Laden Sie sich den Sourcecode für die Übung 1 von der Lernplattform und binden Sie diesen in Ihre IDE ein.

Aufgabe

- Analysieren Sie den Code. Erstellen Sie Komponententests für eine Testabdeckung von 100%. Verwenden Sie jUnit und Mockito. Vergleichen Sie Ihre Lösung mit Ihrem Banknachbarn. Refactoring ist ausdrücklich erlaubt!

Übung 2

Vorbereitung

- Laden Sie sich den Sourcecode für die Übung 2 von der Lernplattform und binden Sie diesen in Ihre IDE ein.

Aufgabe

- Analysieren Sie den Code. Erstellen Sie Komponententests für eine Testabdeckung von 100%. Verwenden Sie jUnit und Mockito. Vergleichen Sie Ihre Lösung mit Ihrem Banknachbarn. Refactoring ist ausdrücklich erlaubt!

Übung 3

Aufgabe

- Erstellen Sie eine Validierungsklasse inkl. Tests mit der Sie folgende Überprüfungen für eine Login-Maske machen können. Eine 100% Testabdeckung ist gefordert!
  - Korrekte E-Mailadresse
  - Passwort ist genügend komplex
    - Mindestens 6 Zeichen
    - Mindestens eine Zahl und eines der folgenden Zeichen «_», «!» oder «#»
  - Passwort ist identisch

Übung 4

Aufgabe

- Erstellen Sie mindestens eine Klasse inkl. Tests die eine Datei einlesen kann. Diese soll darauf reagieren können, wenn die Datei nicht existiert. Eine 100% Testabdeckung ist gefordert!

Übung 5

Aufgabe

- Erstellen Sie eine Klasse, die eine «getValue() : int» Methode aufweist. Versuchen Sie mit Mockito einen Stub von einer Klasse zu erzeugen. Sie dürfen kein Interface verwenden. Was stellen Sie fest?