Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III

Übungen zur Vorlesung Softwaretechnologie

- Wintersemester 2020/20 -

Dr. Günter Kniesel

Übungsblatt 13

Zu bearbeiten bis: 17.01.2020, 16 Uhr

Dieses Blatt wird nicht mehr bepunktet. Manche der Themen darin werden erst nächste Woche angesprochen. Die theoretischen Aufgaben sollen Ihnen bei der Prüfungsvorbereitung helfen. Die angegebenen Punkte sind nur als Idee für eine mögliche Bepunktung ähnlicher Aufgaben in der Klausur zu sehen.

Bitte fangen Sie **frühzeitig** mit der Bearbeitung an, damit wir Ihnen bei Bedarf helfen können. Checken Sie die Lösungen zu den Aufgaben bitte in Ihr Repository ein, „Erklärungen" bitte als Textdatei. Fragen zu Übungsaufgaben respektive zur Vorlesung können Sie auf der Mailingliste swt-tutoren@lists.iai.uni-bonn.de, bzw. swt-vorlesung@lists.iai.uni-bonn.de stellen.

Praktische Aufgaben

Aufgabe 1. *Test-Driven Development* (8 Punkte)

Da die Klasse `GemaltoParser` fehlerhaft aber nicht als Quellcode verfügbar ist, müssen Sie selbst eine korrekte Implementierung der Spezifikation aus Aufgabe 2 von Blatt 12 erstellen. Dabei nutzen Sie die von Ihnen für Aufgabe 12-2 schon erstellten Tests. Gehen Sie wie folgt vor:

- Erzeugen Sie eine Klasse `MyDateParser`, die das Interface DateParser aus Aufgabe 12-2 erst mal durch eine Methode mit leerem Rumpf implementiert.
- Erzeugen Sie eine Klasse `MyDateParserTest`, welche von `GemaltoDateParserTest` erbt, aber die `setUp`-Methode so überschreibt, dass die Variable `systemUnderTest` mit einer Instanz der Klasse `MyDateParser` initialisiert wird.
- Lassen Sie Ihren neuen Test laufen, um zu prüfen, dass das setup schon mal richtig ist. Alle Tests sollten erst mal fehlschlagen.
- Implementieren Sie nun schrittweise die Klasse `MyDateParser`, so dass sie die Spezifikation aus Aufgabe 12-2 erfüllt.

- Übernehmen Sie zuerst das nachfolgend vorgegebene Skelett der Klasse `MyDateParser` und Vervollständigen Sie die parseDate() Method so, dass sie e-minimal funktionsfähig ist, d.h. überhaupt ein Date-Objekt liefert, ohne weitere Überprüfung der Eingabe.

- Lassen Sie Ihre Tests laufen und schauen Sie, welche jetzt erfolgreich sind und welche nicht. Falls gar keine erfolgreich sind, haben Sie vielleicht keinen Test für den „Normalfall“ in Ihrer Suite. Fügen Sie gegebenenfalls einen hinzu.

- Machen Sie nun iterativ weiter, indem Sie sich jeweils einen der noch fehlschlagenden Tests vornehmen und die entsprechend noch nicht erfüllten Anforderungen implementieren und anschließend den JUnit-Test erneut ausführen, bis alle Tests erfolgreich sind.

Zur Dokumentation Ihrer Vorgehensweise checken Sie jedes Mal, wenn weitere Tests Erfolg haben Ihren neuen Stand in Ihr Repository ein.

```java
import org.junit.Before;

public class MyDateParserTest extends GemaltoDateParserTest {

	@Before
	public void setUp() {
		systemUnderTest = new MyDateParser(); // 0,5 Punkte
	}
}
```

```java
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.regex.Pattern;
import java.util.regex.Matcher;


public class MyDateParser implements DateParser {

	Pattern parsePattern =
		Pattern.compile("(\\d{1,2})-(\\d{1,2})-(\\d{2}|\\d{4})");

	@Override
	public Date parseDate(String input) {

		// 1 Punkt: Behandlung von null-Eingabe
		if (input == null)
			return null;

		Matcher dateMatcher = parsePattern.matcher(input);

		if (dateMatcher.matches()){
			int day =   Integer.parseInt(dateMatcher.group(1));
			int month = Integer.parseInt(dateMatcher.group(2));
			int year =  Integer.parseInt(dateMatcher.group(3));

			// 1 Punkt:  „zweistellige Angaben beziehen sich auf das 21.-te  Jahrhundert“
			if (year > 0 && year < 100) {
				year += 2000;
			}
```

```
                    // 1 Punkt: "Wenn Monat kleiner 1 ist, wird Januar angenommen, …"
                    if (month < 1)
                          month = 1;
                    else  // 1 Punkt: "…  bei Werten über 12 Dezember."
                        if (month > 12)
                            month = 12;

                    // 0,5 Punkte: Im GregorianCalendar entspricht 0 Januar:
                    month--;

                    // 1 Punkt: "Wenn Tag kleiner 1 ist wird der Monatserste angenommen, …"
                    if (day < 1)
                          day = 1;

                    // 2 Punkte: „… bei unzulässig großen Tageswerten wird der Monatsletzte angenommen."
                    int[] maxDay = {31,28,31,30,31,30, 31,31,30,31,30,31};
                    final int feb = java.util.Calendar.FEBRUARY;
                    if (month == feb && new GregorianCalendar().isLeapYear(year))
                                maxDay[feb] = 29;
                    if (day > maxDay[month])
                          day = maxDay[month];

                    // 0,5 Punkte: Die Uhrzeit soll 12:00 Uhr sein:
                    GregorianCalendar date =
                                new GregorianCalendar(year,month,day,12,0,0);

                    // 0,5 Punkte: Datumsobjekt zurückgeben:
                    return date.getTime();
                }
                return null;
        }
}
```

Theoretische Aufgaben

Aufgabe 2. *Testen* (→ Kap. 10a) **14 Punkte**

a) **(1,5 Punkte)** Welche grundlegenden drei Arten von Tests gibt es?
→ Folien 9 bis 11: Modul, Integration, System

b) **(1,5 Punkte)** Erklären Sie, was *Akzeptanztests* sind, in welche der obigen drei Kategorien sie gehören, und von wem sie durchgeführt werden.
→ Folie 44: Hilfreich im realen Betrieb, Systemtest, Endbenutzer

c) **(2 Punkte)** Beschreiben Sie den Test-Workflow (textuell oder als Diagramm).
→ Folie 8

d) **(4 Punkte)** Welche Arten des Integrationstests gibt es? Erläutern Sie diese genauer.
→ Folie 33 bis 38: Bottom-up, Top-down, Sandwich

e) **(5 Punkte)** Nennen und erklären Sie die verschiedenen Arten von Performancetests.
→ Folie 43: Timing, Volumen, Stress

Aufgabe 3. *Automatisiertes Testen* (→ Kap. 10b) **18,5 Punkte**

a) **(2 Punkte)** Nennen Sie vier Probleme beim naiven Testen mit Print-Anweisungen.
→ Folie 4: Ständige Programmänderungen, langwierige Testauswertung, Unzuverlässigkeit, Fehleranfälligkeit

b) **(2 Punkte)** Welche Vorteile bringen automatische Tests?
→ Folie 5: Geringer Aufwand, Zuverlässigkeit, schnellere Fehlerentdeckung, schnellere Programmentwicklung

c) **(1 Punkt)** Wie testet man, dass spezifizierte Exceptions tatsächlich geworfen werden in JUnit 3?
→ Folie 14, 19: fail-Assertion nach Testaufruf und try-catch das die korrekte Exception abfängt

d) **(2 Punkte)** Welche Nachteile bringt JUnit 3 mit sich?
→ Folie 21: Tests in getrennter Klasse, Zwang von TestCase zu erben, Namenskonventionen, Aufwendige Suite-Erstellung

e) **(2 Punkte)** Erklären Sie kurz, was Java-Annotationen sind.
→ Folie 24, 25: Annotationen sind strukturierte, selbst-definierbare Modifier, die es ermöglichen Meta-Informationen über Programmelemente bereitzustellen in einer Form die maschinell interpretierbar ist. Sie werden zur Übersetzungszeit vom Java-Compiler oder externen Werkzeugen genutzt, um zusätzliche Checks durchzuführen oder Code zu generieren. Zur Laufzeit werden sie für alles genutzt was via Reflektion möglich ist, unter anderem für automatisiertes Testing

f) **(2,5 Punkte)** Nennen Sie fünf in JUnit 4 verwendete Annotationen und erklären Sie deren Bedeutung.
→ Folie 27: @Test, @BeforeClass, @Before, @After, @AfterClass

g) **(1 Punkt)** Erklären Sie, was Assumptions in JUnit 4 sind.
→ Folie 30, 31: Machen Annahmen unter denen Tests geschreiben werden explizit und vermeiden somit falsche Testergebnisse in Umgebungen, wo die Annahmen nicht gelten

h) **(2 Punkte)** Was ist Spock? Welchen Vorteil hat es gegenüber JUnit?
→ Folie 44-47: DSL für Testing mit Spracherweiterungen `when`, `then` und `expect` die auf den normalen Bedingungen von Java aufbauen. Keine extra Assertions.

i) **(1 Punkt)** Erklären Sie den Audruck „expect“ im Kontext von Spock.
→ Folie 47: Vereinfachung zum Testen von (seiteneffektfreien) Funktionen

j) **(1 Punkt)** Wozu dient Cucumber?
→ Folie 48, 49: Funktionales Testing durch Endanwender

k) **(2 Punkte)** Nennen Sie Gründe die dafür sprechen, Modul-Tests *vor* der Implementierung einer Klasse zu schreiben
→ Folie 53: Test-Driven-Development, Arbeitserleichterung

Aufgabe 4. *Softwareentwicklungsmodelle* (→ Kap. 11) **8 Punkte**

Beschreiben Sie die folgenden Modelle so detailliert, dass die Unterschiede zwischen den jeweiligen Modellen klar werden.

a) **(2 Punkte)** Wasserfall-Modell
→ Folie 14,15:
- (1P) Sequenzielle Ausführung **aller** Aktivitäten einer Kategorie bevor man zur nächsten übergeht.
- (1P) Bei Fehlern wird die aktuelle Phase abgebrochen und zu der zurückgekehrt, aus der der Fehler stammt.

b) **(2 Punkte)** Spiral-Model
→ Folie 23-27:
- (0,5) Der Prozess wird von der Abwägung und Prioritisierung von Risken gesteuert.
- (0,5) Es wird nach abnehmender Risikopriorität (0,5)je ein Prototyp entwickelt, der zeigt, dass (und wie) das jeweilige Risiko bewältigt werden kann.
- (0,5) Anschlließend wird das Ergebnis und das während der Prototyperstellung gelernte reflektiert und fließt in die Neubewertung der Risiken und Planung des nächsten Prototyps ein.
- (0,5) Jeder Prototyp wird entsprechend des Wasserfall-Modells erstellt.

c) **(2 Punkte)** Issue-Based-Modell
→ Folie 34:
- (0,5) Beschreibung des Systems durch Sammlung von offenen/geschlossenen Problemen (‚issues'), …
- … (0,5) die voneinander abhängen können.
- (0,5) Geschlossene Probleme können bei Bedarf wieder geöffnet werden …
- … (0,5 Bonus) z.B. wenn Fehler auftreten, oder Anforderungen sich ändern.
- (0,5) Verschiedene Aktivitäten werden entsprechend der Abhängigkeiten hintereinander ausgeführt, nicht wie beim Wasserfallmodell als Blöcke.

d) **(2 Punkte)** Agile Softwareentwicklung
→ Kap. 12a, Folie 2ff:
- (0,5) Laufende Zusammenarbeit mit Kunden, statt einmalige Vertragsverhandlung
- (0,5)Direkte Reaktion auf Änderungen ((0,5)durch vielfältige Feedback-Zyklen), statt starre Planerfüllung
- (0,5)Fokus auf funktionierender Software statt umfangreicher (aber schnell veralteter) Dokumentation

Aufgabe 5. *Agile Softwareentwicklung* (→ Kap. 12a) **9,5 Punkte**

a) **(1,0 Punkte)** Erklären Sie den Kerngedanken des eXtreme Programming (XP).

→ Folie 3: „Bekannte Prinzipien und Praktiken extrem konsequent umsetzen."

b) **(2,5 Punkte)** Nennen Sie 5 der 7 XP-spezifischen Ausprägungen bekannter Softwareentwicklungs-Prinzipien, die in der Vorlesung besprochen wurden:
→ Folie 8: Kunde ist Teil des Teams, Pair Programming, permanentes Testen, kontinuierliche Integration, Refactoring, einfache Lösungen vorziehen, planning game.

c) **(1 Punkt)** Was bedeutet „Continuous Integration“?
→ Folie 7: Integriere so oft wie möglich! (aber nur wenn alle Tests erfolgreich waren)

d) **(2 Punkte)** Erklären Sie das Konzept des Planning Games.
→ Folie 12-17: Kunden bestimmen Funktionalitäten (Stories) und Prioritäten, Programmierer schätzen jeweils Kosten und Risiken → zusammen festlegen welche Funktionalitäten in der nächsten Iteration realisiert werden sollen, wobei die Kunden über den Funktionsumfang entscheiden.

e) **(2 Punkte)** Wie wirkt sich der Funktionsumfang auf die Qualität und den Zeitrahmen aus?
→ Folie 26, 27: Weniger Funktionen → bessere Qualität im gegebenen Zeitrahmen

f) **(1 Punkt)** Was sind Burn Down Charts?
→ Folie 47-49: Fortschritt / Verzug / Vorarbeit des Projekts zeigen durch Vergleich von realen und idealen Restaufwand → reale Project Velocity vs. konstante Project Velocity

Die Teilaufgabe 5a) wurde gegenüber der ursprünglichen Aufgabenstellung der Klarheit halber in a) und b) aufgeteilt, unter entsprechender Aufteilung der Punkte. Dementsprechend haben sich die restlichen Teilaufgaben verschoben.