Agile Softwareentwicklung mit SCRUM Refactoring 2 with M. Fowler

Version 0.1

13.02.2021

# 

# Dokumentenversionen

|  |  |  |  |
| --- | --- | --- | --- |
| Versionsnr. | Datum | Autor | Änderungsgrund / Bemerkungen |
| 0.2 | 13.03.2021 | Gruppe G | Portfolio-Prüfung SE II |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Inhalt

[Dokumentenversionen 1](#_Toc64108250)

[Einleitung 3](#_Toc64108251)

[Allgemeines 3](#_Toc64108252)

[Epic 1: Projektplanung mit Meilensteinen und Definition von Artefakten 3](#_Toc64108253)

[User-Story 1: 3](#_Toc64108254)

[Epic 2: Refactorings entsprechend der Vorgaben von Martin Fowler 5](#_Toc64108255)

[User-Story 2: 5](#_Toc64108256)

[User-Story 3: 5](#_Toc64108257)

[User-Story 4: 5](#_Toc64108258)

[User-Story 5: 5](#_Toc64108259)

[User-Story 6: 5](#_Toc64108260)

# Einleitung

Diese Projektbeschreibung ist Teil der Portfolio-Prüfung des Moduls Software Engineering II an der DHBW Karlsruhe. Die Prüfungsleistung als Portfolio enthält zwei Workshops zum Nachweis der Lerninhalte aus den Semestern 3 und 4, sowie Grundlagen aus dem Modul Software Engineering I.

### Allgemeines

In diesem Dokument werden Anforderungen für eine agile Umsetzung des Refactoring-Projekts in Epics und User-Stories mit SCRUM vorgegeben.

# Epic 1: Projektplanung mit Meilensteinen und Definition von Artefakten

Projektumsetzung mithilfe agiler Methoden und zugehöriger Artefakte

* Backlog muss vorliegen
  + User-Stories sollten in Sprints verteilt werden (Sprint Planning)
  + User-Stories sollten in entsprechende Tasks eingeteilt werden
  + Tasks sollten nachweislich umgesetzt werden (siehe pushes)
* Durchführung eines Backlog-Groomings pro Sprint
  + Hierzu gehört auch die Bestimmung der Story-Points (ggf. Story Point Poker?)
* Durchführung eines Sprint plannings vor Beginn eines jeden Sprints
* Sprint-Review sollte durchgeführt werden
* Github o.ä. um Code-Iterationen (pushes) zu erkennen (sollten zu User-Stories, bzw. den daraus abgeleiteten Tasks passen)

Die hierfür erforderliche Arbeit eines Product-Owners wurde schon teilweise abgenommen, dies beinhaltet:

* Erstellen relevanter User-Stories aus dem entsprechenden Haupt-Epic
* Priorisieren der User-Stories nach diversen Faktoren (Bearbeitungs-, bzw. Relevanz-Priorisierung wird durch die Nummerierung, bzw. Reihenfolge der Refactoring-User-Stories vorgegeben)

Folgende Punkte sind für die vorhandenen User-Stories noch zu erledigen (Definition of Ready):

* Story-Point-vergabe je User-Story
* Verständnis im Team
* DoD geklärt?

### User-Story 1:

Als „Software Quality Engineer“ möchte ich, dass sich das Entwicklungsteam im zu bearbeitenden Coding auskennt, um eine hohe Entwicklungsqualität und -geschwindigkeit zu gewährleisten.

* Beispielstask: Zusammensetzen und Code (IST-Stand) gemeinsam analysieren.
* Definition of Ready:
  + Definition of Done geklärt (Projektübergreifend)
  + Story-Points vergeben (Grooming)
  + jedem Teammitglied ist klar, dass es darum geht, den Code zu verstehen
  + In Task(s) unterteilt -> z.B. Einrichten der Entwicklungsumgebung + erstes öffnen des Projekts (jeder im Team) + gemeinsame Analyse / selbstständige Analyse etc.

### Definition of Done:

Der Code ist geschrieben.

Der Code ist getestet.

Es wurde eine Code-Review durchgeführt.

# Epic 2: Refactorings entsprechend der Vorgaben von Martin Fowler

Die für das Refactoring des Beispiel-Projekts aufgelisteten User-Stories sind exemplarisch und erheben insbesondere keinen Anspruch auf Vollständigkeit. Ziel ist es analog zum Fowler-Vortrag der letzten Woche sinnvolle Arbeitsschritte zu identifizieren und diese im Folgenden in Form von User-Stories abzubilden. Da Refactoring ein iterativer Prozess ist ist es sinnvoll die gewünschten Refactorings Schritt für Schritt zu spezifizieren und umzusetzen.

Grundlage zur Identifikation notwendiger Refactorings sind die vom Kunden gewünschten Erweiterungen der Software:

* Außer der Plan-Text Aufstellung für die Rechnungen sollen in Zukunft auch HTML-Ausgaben unterstützt werden.
* In Zukunft werden neben comedy und tragedy auch Stücke der Typen history, pastoral, pastoral­comical, historical­pastoral, tragicalhistorical, tragical­comical­historical­pastoral, scene individable und poem unlimited in das Programm aufgenommen. Diese Aufstellung ist nicht abschließend bzw. vollständig und wann die jeweilige Erweiterung notwendig wird ist nicht klar.

### User-Story 2:

Als „Software Quality Engineer“ möchte ich, dass die switch-Anweisung zur Unterscheidung von Typen bei der Berechnung von Aufführungen in eine eigene Methode ausgelagert wird, sodass hierdurch die Übersichtlichkeit und Wartbarkeit des Codes gesteigert wird (Replace Type Code with State/Strategy).

Tasks:

* Amount-Berechnung in eigene Methode auslagern (entsprechende Benennung)
* Aufruf der neuen Methode in altem Code
* Interne Variablen der extrahierten Methode umbenennen

### User-Story 3:

Als „Software Quality Engineer“ möchte ich, dass sich die Methode zur Preisberechnung in der hierzu geeigneten Klasse befindet, sodass die Kohäsion des Codes erhöht wird (Move Method).

Tasks:

* Methode in neue PerformanceCalculator-Klasse verschieben
* Methodenaufruf anpassen

### User-Story 4:

Als „Software Quality Engineer“ möchte ich, dass temporäre Variablen vermieden und die Methode VolumeCredits() soweit möglich als Query aufgerufen wird, sodass die Code-Menge reduziert wird (Replace Temp with Query).

Tasks:

* VolumeCredits-Berechnung in eigene Methode auslagern
* Aufruf der neuen Methode in altem Code
* Interne Variablen der extrahierten Methode umbenennen

### User-Story 5:

Als „Software Quality Engineer“ möchte ich, dass die Variablen verständliche und intuitive Namen haben, um den Code verständlicher zu machen, beziehungsweise nachvollziehbarer zu gestalten.

Tasks:

* Variablen passend umbenennen

### User-Story 6:

Als „Software Quality Engineer“ möchte ich, dass NumberFormat.format() in eine eigene Methode ausgelagert wird, um der Refactoring-Methode „Extract Method“ gerecht zu werden und den Code so zu verbessern.

Tasks:

* usd() Methode implementieren (statt NumberFormat.format())
* die Methode in print() verwenden

### User-Story 7:

Als „Software Quality Engineer“ möchte ich, dass die for-Schleife in Print in zwei neue Methoden geteilt wird, um die Code-Qualität, sowie dessen Modularität zu erhöhen.

Tasks:

* for-Schleife in der Methode print() aufteilen
* jeweils „entstehende“ for-Schleife in neue Methode totalVolumeCredits() und totalAmounts() implementieren
* die jeweilige Methode in der print() Methode verwenden

### User-Story 8:

Als „Software Quality Engineer“ möchte ich, dass die switch-case-Anweisung aus PerformanceCalculator in Polymorphie umgebaut wird, um dem Refactoring „Replace Switch Polymorphism“ gerecht zu werden.

Tasks:

* für jede Kategorie der Stücke eine neue Klasse erstellen, die von PerformanceCalculator erbt
* in den Subklassen jeweils die Methoden amount() und volumeCredits() wie notwendig überschreiben
* neue Methode in Klasse PerformanceCalculator einführen, die gemäß dem Architekturmuster „Factory“ eine neue Instanz der gewünschten Subklasse zurückgibt

### User-Story 9:

Als „Software Quality Engineer“ möchte ich, dass die print() Methode in eine neue Klasse ausgelagert wird und dort gemäß der „Form Template Method“ strukturieren, um den Code für neue Ausgabenvarianten offen zu machen (Open-Closed-Principle).

Tasks:

* neue abstrakte Klasse Statement erstellen
* in der Klasse eine Methode value() erstellen
* neue abstrakte Methode header() erstellen
* neue abstrakte Methode position() erstellen
* neue abstrakte Methode footer() erstellen
* abstrakte Methoden in der value() Methode verwenden
* neue Subklasse PlainTextStatement von Statement erstellen
* in PlainTextStatement die abstrakten Methoden header(), position() und footer() ausprogrammieren

### User-Story 10:

Als „Product Owner“ möchte ich, dass eine neue Statement-Subklasse eingeführt wird, die eine HTML-Ausgabe generiert, um die Software für weitere Verarbeitung gestalterisch auszubauen.

Tasks:

* Neue Subklasse von Statement
* HTML-Anweisung ausprogrammieren

### User-Story 11:

Als „Software Quality Engineer“ möchte ich, dass in der Klasse Performance die Referenz auf das Play-Objekt über die playID in eine direkte Objekt-Referenz geändert wird, um objektorientiertes Refactoring zu ermöglichen.

Tasks:

* Variable playID von String auf Play ändern und Konstruktor anpassen
* JUnit Tests anpassen (Map durch Liste ersetzen; Konstruktoraufruf von Performance anpassen)

### User-Story 12:

Als „Software Quality Engineer“ möchte ich, dass die Methoden aus der StatementPrinter Klasse in die jeweilige Klasse bewegt werden.

Tasks:

* Methode amount() in Performance
* Methode volumeCredits() in Performance
* Methode totalAmount() in Invoice
* Methode totalVolumeCredits() in Invoice
* Neue Methode getPlay() in Performance

### User-Story 13:

Als „Software Quality Engineer“ möchte ich, dass temporäre Variablen durch Methodenaufrufe ersetzt werden, um dem Refactoring-Prinzip „Replace Temp with Query“ gerecht zu werden.

Tasks:

* Anpassungen in print() in StatementPrinter
* Anpassungen in getTotalAmountFor() in StatementPrinter
* Anpassungen in getTotalVolumeCreditsFor() in StatementPrinter

### User-Story 14:

Als „Software Quality Engineer“ möchte ich, dass die Klasse Play abstrakt wird und eine Subklasse für jede Kategorie erstellt wird.

Tasks:

* Play abstrakt machen
* Subklassen erstellen
* Factory Methode von PerformanceCalculator auf Subklassen aufteilen