Agile Softwareentwicklung mit SCRUM Refactoring 1 with M. Fowler

Version 0.3

13.02.2021

# 

# Dokumentenversionen

|  |  |  |  |
| --- | --- | --- | --- |
| Versionsnr. | Datum | Autor | Änderungsgrund / Bemerkungen |
| 0.1 | 02.06.2020 | Dr. Marc Schanne | Ersterstellung |
| 0.2 | 20.06.2020 | Gennaro Izzo | Review |
| 0.3 | 13.02.2021 | Dr. Marc Schanne | 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?

Aufbau einer User-Story:

Als [Nutzer / Rolle] möchte ich [was?], dass / weil / um [warum? / Nutzen / Wert].

### 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.

# Epic 2: Refactorings entsprechend der Vorgaben von Martin Fowler

Die in den Folien von Martin-Fowler beschriebenen Arbeitsschritte werden im Folgenden in Form von User-Stories abgebildet (siehe Anforderung 1).

Zur DoD zählt auch immer, dass nach Erledigung der User-Stories alle Tests noch funktionieren.

### User-Story 2: 2 Story points

Als „Software Quality Engineer“ möchte ich, dass die switch-Anweisung zur Preisberechnung in eine eigene Methode ausgelagert wird, sodass hierdurch die Übersichtlichkeit und Wartbarkeit des Codes gesteigert wird.

Tasks:

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

DoD:

* Amount-Calculation in eigene Methode ausgelagert
* Methode wird im alten Code aufgerufen
* Interne Variablen der extrahierten Methode umbenannt

### User-Story 3: 1 Story Point

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.

Tasks:

* Methode in Rental-Klasse verschieben
* Methodenaufruf anpassen

DoD:

* Methode wurde in Rental-Klasse verschoben
* Methodenaufruf wurde angepasst

### User-Story 4: 5 Story Points

Als „Software Quality Engineer“ möchte ich, dass temporäre Variablen vermieden und die Methode zur Preisberechnung soweit möglich als Query aufgerufen wird, sodass die Code-Menge reduziert wird.

Tasks:

* getTotalCharge in Methode auslagern und als Query berechnen lassen
* getTotalFrequentPoints in Methode auslagern und als Query berechnen lassen

DoD:

* getTotalCharge wurde in Methode ausgelagert und wird als Query berechnet
* getTotalFrequentPoints wurde in Methode ausgelagert und wird als Query berechnet

### User-Story 5: 1 Story Point

Als „Software Quality Engineer“ möchte ich, dass die Berechnung der frequentRenterPoints in die relevante Klasse ausgelagert (Kombination aus Extract und Move Method), um die Kohärenz zu reduzieren und um Separation of Concerns zu gewährleisten.

Tasks:

* Methode in entsprechende Klasse auslagern
* Methode an alter Stelle aufrufen

DoD:

* Methode wurde in entsprechende Klasse ausgelagert
* Neue Methode wird an alter Stelle aufgerufen

### User-Story 6: 3 Story Points

Als „Software Quality Engineer“ möchte ich, dass die temp-Variablen „totalAmount“ und „frequentRenterPoints“ aus dem Code entfernt werden, um die Lesbarkeit des Codes zu verbessern.

Tasks:

* Temp-Variablen entfernen und durch Methodenaufrufe ersetzen

DoD:

* Temp-Variablen wurden entfernt und durch Methodenaufrufe ersetzt

### User-Story 7: 5 Story Points

Als „Nutzer“ möchte ich, dass die Ausgabe als html-Statement erfolgt.

Tasks:

* Ausgabe in html-Statement umwandeln

DoD:

* Ausgabe wurde in html-Statement umgewandelt

### User-Story 8: 2 Story Points

Als „Software Quality Engineer“ möchte ich, dass die Methoden getCharge() und frequentRenterPoints() in die Klasse Movie ausgelagert werden, um das Single Responsibility Pattern zu beachten.

Tasks:

* getCharge() in Movie auslagern und Fallunterscheidung Children, Regular und New\_Release
* frequentRenterPoints() in Movie auslagern
* Methoden an alter Stelle aufrufen

DoD:

* GetCharge() wurde in Movie ausgelagert
* Dabei wurde Fallunterscheidung vorgenommen
* FrequentRenterPoints() wurde in Movie ausgelagert
* Methoden werden an alter Stelle aufgerufen

### User-Story 9: 4 Story Points

Als „Software Quality Engineer“ möchte ich, dass die Fallunterscheidung von User-Story 8 in der Klasse Movie durch Polymorphie umgesetzt werden, da der Code dadurch sowohl lesbarer als auch übersichtlicher wird. Zudem ist switch-case auch immer ein Codesmell.

Tasks:

* Unterklassen erstellen
* Methoden definieren
* Methoden überschreiben in den Unterklassen

DoD:

* Unterklassen sind erstellt
* Methoden in der Oberklasse sind definiert
* Methoden werden in den Unterklassen entsprechend überschrieben

### User-Story 10: 11 Story points

Als „Software Quality Engineer“ möchte ich, dass die Statements in Unterklassen ausgelagert werden, da damit das Strategy-Pattern umgesetzt wird. Dadurch ist der Anwender freier in der Entscheidung der Art des Ergebnisses (HTML oder normaler Text).

Tasks:

* Interface Statement implementieren
* Klasse HTML Statement implementieren
* Klasse Text Statement implementieren
* Aufrufe in Customer implementieren

DoD:

* Interface Statement implementiert
* Klasse HTML Statement implementiert
* Klasse Text Statement implementiert
* Aufrufe in Customer implementiert

**Ab hier sollten die Studenten die User-Stories selbstständig beschreiben.**