Agile Softwareentwicklung mit SCRUM Refactoring 2 with M. Fowler

Version 0.1

13.02.2021

# 

# Dokumentenversionen

|  |  |  |  |
| --- | --- | --- | --- |
| Versionsnr. | Datum | Autor | Änderungsgrund / Bemerkungen |
| 0.1 | 13.02.2021 | Dr. Marc Schanne | Portfolio-Prüfung SE II |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Inhalt

[Dokumentenversionen 1](#_Toc66526746)

[Einleitung 3](#_Toc66526747)

[Allgemeines 3](#_Toc66526748)

[Epic 1: Projektplanung mit Meilensteinen und Definition von Artefakten 3](#_Toc66526749)

[User-Story 1: 3](#_Toc66526750)

[Epic 2: Refactorings entsprechend der Vorgaben von Martin Fowler 5](#_Toc66526751)

[User-Story 2: 5](#_Toc66526752)

[User-Story 3: 5](#_Toc66526753)

[User-Story 4: 5](#_Toc66526754)

[User-Story 5: 5](#_Toc66526755)

[User-Story 6: 6](#_Toc66526756)

[Definition of Done 6](#_Toc66526757)

# 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.

Link zum Kanban-Board bei trello.com: <https://trello.com/invite/b/ORqdVWHE/41796b7bf999eed7d5e23faf791d23da/scrum-entwicklung>

# 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 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.

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.

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 zur Preisberechnung soweit möglich als Query aufgerufen wird, sodass die Code-Menge reduziert wird.

### User-Story 5:

Als „Buchhalter“ möchte ich mir Abrechnungen und Rabattoptionen sowohl als Text als auch in HTML-Form anzeigen lassen, um eine flexiblere Anzeige zu haben.

### User-Story 6:

Als „Software Quality Engineer“ möchte ich, dass für die HTML-Ausgabe des Statements ein Test exisitiert.

# Definition of Done

* Feature fertig
* Code fertig
* keine bekannten Fehler
* Abgenommen durch den Product Owner
* Bereit für die Übernahme in die Produktivumgebung