Agile Softwareentwicklung mit SCRUM Refactoring the Core War

Version 0.3

27.06.2020

# 

# Dokumentenversionen

|  |  |  |  |
| --- | --- | --- | --- |
| Versionsnr. | Datum | Autor | Änderungsgrund / Bemerkungen |
| 0.1 | 02.06.2020 | Dr. Marc Schanne | Ersterstellung |
| 0.2 | 27.02.2020 | Gennaro Izzo | Review |
| 0.3 | 27.06.2020 | Melise Akcay (Gruppe I) | Ergänzung weiterer Stories |
|  |  |  |  |

Inhalt

[Dokumentenversionen 1](#_Toc44164115)

[1. Einleitung 3](#_Toc44164116)

[1.1. Allgemeines 3](#_Toc44164117)

[1.1.1. Ziel und Zweck dieses Dokumentes 3](#_Toc44164118)

[1.1.2. Projektbezug 3](#_Toc44164119)

[1.1.3. Anmerkung zu Prozess- und Projekt 3](#_Toc44164120)

[2. Projektanforderungen in Epics 3](#_Toc44164121)

[Epic 1: Projektplanung mit Meilensteinen und Definition von Artefakten 4](#_Toc44164124)

[User-Story 1: Analyzation of current state 5](#_Toc44164125)

[Epic 2: Refactorings entsprechend eigener Analysen 5](#_Toc44164126)

[User-Story 1: Seperation of Concerns 1 5](#_Toc44164127)

[User-Story 2: Seperation of Concerns 2 5](#_Toc44164128)

[User-Story 3: run application without arguments 6](#_Toc44164129)

[User-Story 4: Seperation of Concerns 3 6](#_Toc44164130)

[Epic 3: Dokumentation entsprechend der Refactorings anpassen 6](#_Toc44164131)

[User-Story 1: Documentation of Refactoring 6](#_Toc44164132)

[User-Story 2: Handbook for the User 7](#_Toc44164133)

# Einleitung

Diese Spezifikation ist Teil der Prüfung des Moduls Software Engineering II im Jahrgang WWI17 an der DHBW Karlsruhe. Die Prüfungsleistung als Portfolio enthält zwei Workshops zum Nachweis der Lerninhalte aus den Semestern 5 und 6, sowie Grundlagen aus dem Modul Software Engineering I.

## Allgemeines

In diesem Dokument werden konkrete Epics mit User Stories und Tasks für die Umsetzung in einem agilen Softwareentwicklungsprozess (SCRUM) vorgegeben.

### Ziel und Zweck dieses Dokumentes

Ziel dieser Spezifikation ist es die Anforderungen des Auftraggebers (AG) an das vorliegende Refactoring-Projekt in Epics und User Stories festzuhalten, damit der Product Owner (PO) des Auftragnehmers (AN) auf dieser Grundlage konkrete Tasks identifizieren kann, damit diese für die weitere Entwicklung priorisiert und im Backlog verwaltet werden können.

### Projektbezug

Ziel dieses Softwareentwicklungs- und Refactoring-Projektes ist es eine Java-Implementierung des Programmierspiels Core War (Krieg der Kerne) durch ein umfangreiches Refactoring zu renovieren und für die erfolgreiche Nutzung auf dem modernen Desktop aufzubereiten.

### Anmerkung zu Prozess- und Projekt

Dieses Dokument muss im ersten Schritt des Softwareentwicklungsprozesses durch den PO und das Entwicklerteam (DEV) erweitert (User Stories für gegebene Epics) und detailliert (Tasks) werden.

Die notwendige Priorisierung der identifizierten User Stories nach diversen Faktoren (Bearbeitungs-zeit, bzw. Relevanz) wird durch die Nummerierung, bzw. Reihenfolge der User Stories vorgegeben.

Für alle User Stories (auch die vorgegebenen) sind die folgenden Punkte obligatorisch durchzuführen, bzw. zu vervollständigen:

* Vergabe von Story Points
* Verständnis im Team
* Klärung der Definition of Done (DoD)

# Projektanforderungen in Epics

Ausgangspunkt für das vorliegende Refacoring-Projekt ist die Implementierung des bekannten Programmierspiels Core War in einem Java Applet aus den Neunziger-Jahren des letzten Jahrtausends.

Core War ist ein Programmierspiel das A. K. Dewdney im Mai 1984 als Zerstreuung für Programmierer erfunden hat. Hintergrund war der Aufstieg von Computer-Viren und der Kampf gegen diese unerwünschten „Programme“ auf modernen Peronal Computern mit Anti-Viren-Programmen.

Dazu hat Dewdney einen virtuellen Computer mit einem eindimensionalen Ringspeicher definiert, in dem zwei oder mehr Programme, die in einer assemblerartigen Sprache (Redcode) geschrieben sind, als Krieger im gemeinsamen Speicher (MARS, Memory Array Redcode Simulator) ausgeführt werden.

Die International Core War Society (ICWS) hat 1984 und 1988 Revisionen von Redcode veröffentlicht und ein Update-Vorschlag von 1994 wurde zwar als Draft veröffentlicht, aber nicht final verabschiedet.

Als Referenzimplementierung für MARS mit einem Redcode-Interpreter (in Teilen bereits für die Version 1994), der die Ausführung der Kämpfer simuliert, wurden über die Jahre auf sourceforge.net in C als portable Memory Array Recode Simulator (pMARS) entwickelt.

Diverse Neuimplementierung (z.B. nMARS) aber auch eine Online-Version unter https://crypto.stanford.edu/~blynn/play/redcode.html sind mittlerweile (noch) verfügbar.

Ausgangspunkt dieses Refactoring-Projekts ist eine Implementierung in Java (jMARS), die als Java-Applet und SWT-Applikation verfügbar war. Für die Nutzung mit modernem Java ist der Code in einem Github Repository als einfache AWT-Applikation restauriert und mit einer Projektstruktur in Apache Maven als Apache NetBeans Projekt aufbereitet worden.



# Epic 1: Projektplanung mit Meilensteinen und Definition von Artefakten

Epic: 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: Analyzation of current state

Als für die Weiterentwicklung verantwortlicher Chefarchitekt möchte ich, dass 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 eigener Analysen

Das Refactoring des Core-War Projekts ist nicht durch eine Folien-Präsentation von Martin Fowler vorgegeben.

Die bekannten Code-Smells sollten bei der Analyse des Codes genutzt werden um sinnvolle Refactorings zu identifizieren. Konkrete Beispiel sind die Weitergabe von Parametern und Ergebnissen über Zustände, die Kapselung von Code und die Struktur des Kontrollflusses allgemein.

### User-Story 1: Seperation of Concerns 1

Als für die Weiterentwicklung verantwortlicher Chefarchitekt möchte ich, dass der bestehende Projektcode überarbeitet wird und die alte Parameterweitergabe wie bei einem Applet entfernt wird.

* Task 1: Im Hauptprogramm corewars.jmars.jMARS wird die Speicherung von Konfigurations-parametern in Instanzvariablen ersetzt.
  + Identifikation relevanter Argumente und Übergabe an die verantwortlichen Methoden.
* Task 2: Im Hauptprogramm corewars.jmars.jMARS wird der Aufruf von Methoden ohne Argumenten durch einen direkten Kontrollfluss ersetzt.
  + Identifikation relevanter Argumente und Übergabe an die verantwortlichen Methoden.

### User-Story 2: Seperation of Concerns 2

Als für die Weiterentwicklung verantwortlicher Chefarchitekt möchte ich, dass der bestehende Projektcode überarbeitet wird und inhaltlich getrennte Aufgaben in verschiedenen Klassen bzw. Methoden umgesetzt werden.

* Task 1: Die Ergebnisausgabe ist Teil des MARS-Verarbeitungsthreads, dies muss entzerrt werden.
  + Ziel ist die lose Kopplung zwischen MARS-Ausführung, der Anzeige und der Ergebnisausgabe. Dies erlaubt später Erweiterungen für eine geänderte Statistik-Ausgabe.
  + Allgemein wird die Kapselung von Code überprüft und evtl. geändert.
* Task 2: Die GUI-Ausgaben müssen in einer eigenen GUI-Klasse ausgelagert werden.
  + Ziel ist die lose Kopplung zwischen MARS-Ausführung und der Ausgabe. Sie erlaubt später die Erweiterung oder Änderung der GUI.
  + Allgemein wird die Kapselung von Code überprüft und evtl. geändert.
* Task 3: Die Konsolen-Ausgaben müssen in einer eigenen Klasse ausgelagert werden.
  + Ziel ist die lose Kopplung zwischen MARS-Ausführung und der Ausgabe. Sie erlaubt später die Erweiterung oder Änderung.
  + Allgemein wird die Kapselung von Code überprüft und evtl. geändert.

### User-Story 3: run application without arguments

Als für die Weiterentwicklung verantwortlicher Chefarchitekt möchte ich, dass der bestehende Projektcode überarbeitet wird und somit die Applikation ohne Angaben von Argumenten ausführbar umgesetzt wird.

* Task 1: Auslagerung der Frame-Erstellung in eigener Methode
  + Ziel ist die lose Kopplung zwischen MARS-Ausführung, der Anzeige und der Ergebnisausgabe. Dies erlaubt später Erweiterungen für eine geänderte Statistik-Ausgabe.
  + Allgemein wird die Kapselung von Code überprüft und evtl. geändert.

### User-Story 4: Seperation of Concerns 3

Als für die Weiterentwicklung verantwortlicher Chefarchitekt möchte ich, dass der bestehende Projektcode überarbeitet wird und inhaltlich getrennte Aufgaben in verschiedenen Klassen bzw. Methoden umgesetzt werden.

* Task 1: Auslagerung der Frame-Erstellung in eigener Methode
  + Ziel ist die lose Kopplung zwischen MARS-Ausführung, der Anzeige und der Ergebnisausgabe. Dies erlaubt später Erweiterungen für eine geänderte Statistik-Ausgabe.
  + Allgemein wird die Kapselung von Code überprüft und evtl. geändert.
* Task 2: Schleife in der applicationInit-Methode auslagern
  + Ziel ist die lose Kopplung zwischen MARS-Ausführung, der Anzeige und der Ergebnisausgabe. Dies erlaubt später Erweiterungen für eine geänderte Statistik-Ausgabe.
  + Allgemein wird die Kapselung von Code überprüft und evtl. geändert.
* Task 3: paser-Initialisierung ausgelagert
  + Ziel ist die lose Kopplung zwischen MARS-Ausführung, der Anzeige und der Ergebnisausgabe. Dies erlaubt später Erweiterungen für eine geänderte Statistik-Ausgabe.
  + Allgemein wird die Kapselung von Code überprüft und evtl. geändert.

# Epic 3: Dokumentation entsprechend der Refactorings anpassen

### User-Story 1: Documentation of Refactoring

Als Product Owner möchte ich eine Word-Datei zur Dokumentation des Projektes, um die Änderungen zu verstehen.

* Task 1: Word-Dokument anpassen
  + Ziel ist es, dem Product Owner einen Überblick über die Refactorings und die umgesetzten User-Stories zu verschaffen.

### User-Story 2: Handbook for the User

Als Benutzer möchte ich ein Benutzerhandbuch, um die Applikation zu verstehen und einfach anzuwenden.

* Task 1: Benutzerhandbuch erstellen
  + Ziel ist es, dem Benutzer eine Einleitung zur Applikation an die Hand zu geben