Projektmanagement-Plan - Message Logger
Team 4

Stephan Stofer
Maurizio Hostettler
Julien Grüter
Adrian Willi

Frühlingssemester 2020
Version 2.0.0

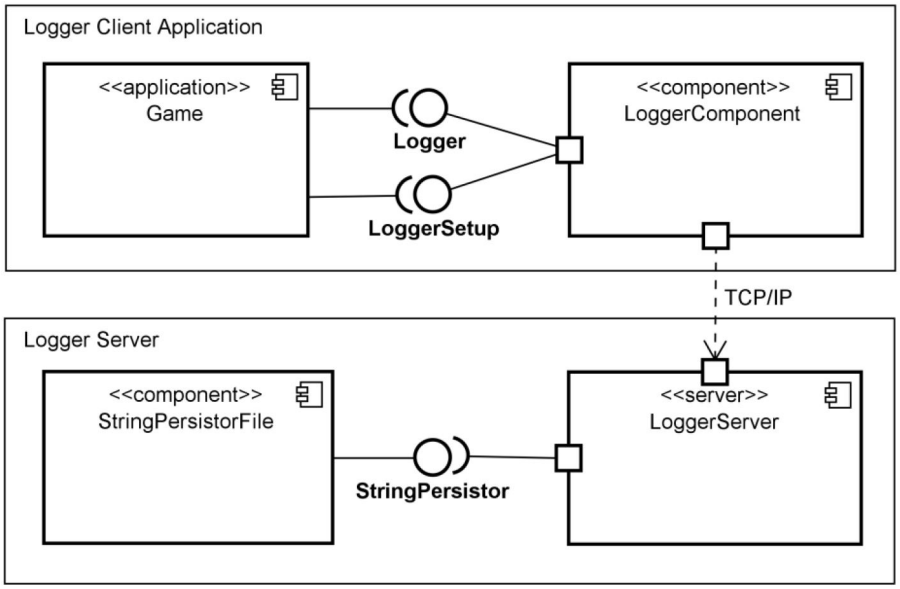
Rev	Datum	Autor	Bemerkungen	Status
1.0.0	19.04.2020	Alle	1. Version - Zwischenabgabe	Fertig
1.0.1	20.04.2020	Alle	Überarbeitete 1. Version gemäss Feedback Zwischenabgabe	Fertig
1.0.2	17.05.2020	Alle	Überarbeitete Version gemäss Review	Fertig
2.0.0	18.05.2020	Alle	Schlussabgabe	Fertig

Contents

1 Projektorganisation

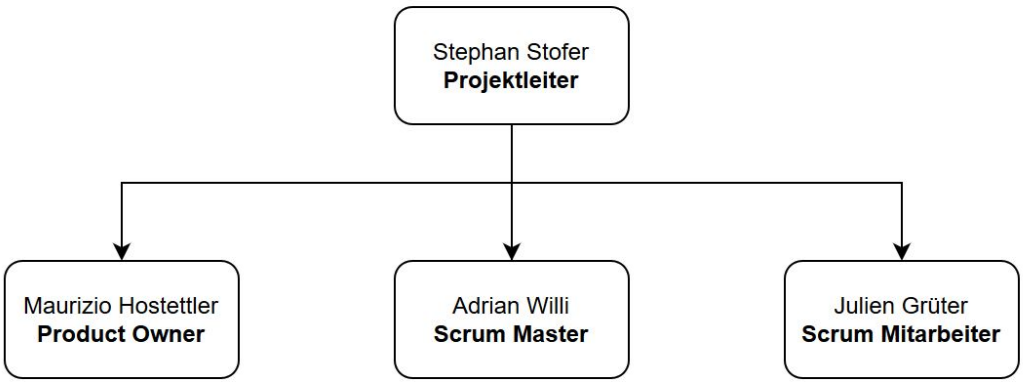
1.1 Projektauftrag

Es soll eine Logger-Komponente für ein Spiel entwickelt werden. Bei der Logger-Komponente handelt es sich um eine Software-Komponente, die einfach in bestehende Java-Applikationen eingebunden werden kann. Eine Applikation kann durch einfache Methodenaufrufe Meldungen (Messages) über die Logger-Komponente dauerhaft aufzeichnen. Die Meldungen aller Applikationen sollen in einem zentralen Logger Server in einem wohldefinierten Format gespeichert werden. Über den LoggerViewer können die geloggten Nachrichten auf dem LogServer in einem GUI angezeigt werden.



1.2 Projektmanagement

1.2.1 Organigramm



1.2.2 Rollen und Zuständigkeiten

- **Stephan Stofer - Projektleiter / Scrum Mitarbeiter:** Der Projektleiter wird in der Initialisierungs- und Einführungsphase eine Schlüsselrolle einnehmen. Der Fokus seiner Tätigkeit wird in der Organisation des Projektes sowie in der Erstellung eines Rahmenplans liegen. Diser Rahmenplan gibt eine grobe Vorstellung des übergeordneten Projektablaufs mit den wichtigsten Meilensteinen und Terminen.
- **Maurizio Hostettler - Product Owner / Member Interface-Komitee / Scrum Mitarbeiter:** Der Product Owner wird in der Konzeptions- und Realisierungsphase eine entscheidende

Rolle einnehmen. Seine Haupttätigkeit umfasst die Product Backlog-Pflege und -Priorisierung sowie die Sprint-Planung und -Abnahme. Als Zusatzfunktion wird er innerhalb der Gruppe noch als Member des Interface-Komitees agieren.

- **Adrian Willi - Scrum Master / Scrum Mitarbeiter:** Der Scrum Master ist in jeder Phase des Projekts essentiell. Er wird als Vermittler zwischen dem Scrum Team und dem Product Owner agieren. Zudem unterstützt er ein korrektes Projektvorgehen und stellt die Qualität der Leistung sicher.
- **Julien Grüter - Scrum Mitarbeiter:** Er wird bei der Entwicklung eine wichtige Rolle spielen und kann dort seine Erfahrungen aus der Praxis einfliessen lassen. Er wird auch sicherstellen, dass der Code den Anforderungen entspricht und andere Teammitglieder bei Bedarf unterstützen bei Ihren Tätigkeiten.

1.2.3 Projektstrukturplan

Auf einen Projektstrukturplan wird explizit verzichtet, da das Projekt klein ist und sonst nur ein unnötiger Overhead entsteht.

2 Projektführung

Das Projektmanagement in diesem Projekt wird mit der Projektmethode SoDa gemacht.

2.1 Rahmenplan

Hierbei handelt es sich um einen aktualisierten Rahmenplan. Die Änderungen wurden ausgelöst durch die Umstellung auf reinen Onlineunterricht.

Wichtige Daten:

- **Meilensteine:**
  - Meilenstein 1: SW03 (02.03.2020)
  - Meilenstein 2: SW10 (20.04.2020)
  - Meilenstein 3: SW15 (25.05.2020)
- **Sprints:**
  - Sprint 1: SW04 - SW05
  - Sprint 2: SW06 - SW09
  - Sprint 3: SW11 - SW12
  - Sprint 4: SW13 - SW14

2.2 Projektkontrolle

Die Projektkontrolle wird im Gitlab geführt und ist dort ersichtlich.

2.3 Meilensteinplanung

Die Meilensteinplanung wurde auf Basis des Rahmenplans erstellt. Das Projekt gliedern wir in 3 Meilensteine. Dabei ist zu Beginn, in der Mitte des Projektes sowie am Ende ein Meilenstein vorgesehen. Die Meilensteine richten sich an den Vorgaben des Auftragsgebers aus. Im Anhang sind die Ziele und Reviews der Meilensteine ersichtlich in Form von Meilensteinberichten.

2.4 Sprintplanung

Die Sprintplanung ist eine rollende Detailplanung. Der Product Owner priorisiert laufend, zusammen mit dem Entwicklungsteam und dem Auftraggeber, die Epic, respektive die User Stories im Backlog. Zusammen mit dem Entwicklungsteam schätzt der Product Owner den Aufwand der höchst priorisierten Anforderungen und setzt diese im folgenden Sprint um. Die Sprintdauer umfasst im Normalfall 2 Wochen, wobei Abweichungen natürlich möglich sind.
Die Aufwandschätzungen in der einzelnen Stories werden jeweils nach Personentagen im GitLab vorgenommen.
Im Anhang sind der Product Backlog, Sprintplanungen, Sprintreviews sowie Retrospektiven ersichtlich.
Die einzelenen Item und Sprints sind ebenfalls im Gitlab ersichtlich.

2.4.1 Definition of Done

```
### Definition of Done
* [ ] Unit Tests sind implementiert und decken min. 80% des Codes ab (Jacoco Report).
* [ ] Code Review durch einen anderen Scrum Mitarbeiter hat stattgefunden.
* [ ] Der Build Prozess konnte erfolgreich durchgeführt werden.
* [ ] Die Dokumentation (Projektdokumentation und JavaDoc) wurden nachgeführt.
```

Diese Definition wird im GitLab bei jedem Issue hinzugefügt und entsprechend angepasst. Speziell werden die spezifischen Akzeptanzkriterien jeweils festgehalten.

2.5 Risikomanagement

Im Rahmen der Risikoanalyse haben wir folgende Risiken identifiziert:

2.5.1 Unvorhergesehene Anforderungsänderungen während eines Sprints

Anforderungen an Komponenten oder Interface ändern sich während Entwicklungszyklen fundamental.

Massnahmen zur Risikominderung

- Falls Probleme oder Unsicherheiten entstehen, sind diese sofort mit den anderen Team-Mitglieder zu besprochen. (Eintrittswahrscheinlichkeit minimieren)
- Pro Sprint einen halben Tag Pufferzeit einplanen. (Eintrittswahrscheinlichkeit minimieren)

2.5.2 Ausfall Projektmitglied

Ausfall eines Projektmitglieds durch Unfall oder Krankheit.

Massnahmen zur Risikominderung

- Regelmässiger Austausch mit den Mitgliedern, um über den jeweils aktuellen Stand zu informieren (Auswirkung minimieren)
- Gitlab sowie die Dokumentation auf dem aktuellen Stand halten (Auswirkung minimieren)
- Gesunde Ernährung mit viel Gemüse und Früchten (Eintrittswahrscheinlichkeit minimieren)
- Ausreichend Schlaf und genügend Pausen während längeren Arbeitsphasen (Eintrittswahrscheinlichkeit minimieren)

2.5.3 Ungleiche Arbeits-/Wissensverteilung in Projektgruppe

Aufgaben und oder Wissen über Projektfortschritte unregelmässig in Projektgruppe verteilt.

Massnahmen zur Risikominderung

- Weekly MeetUp vor oder nach der Vorlesung in dem wir uns über das Projekt, Projektstand sowie Aufgaben während der Woche austauschen (Auswirkung minimieren)
- Sprintplan in Gitlab ist von allen Mitgliedern regelmässig prüfen (Auswirkung minimieren)
- Regelmässiger Austausch mit den Mitgliedern, um über den jeweils aktuellen Stand zu informieren (Auswirkung minimieren)
- Gitlab sowie die Dokumentation auf dem aktuellen Stand halten (Auswirkung minimieren)
- Austausch bei Fragen oder Problemen ausserhalb des Vorlesungsblocks findet über WhatsApp-Gruppenchat statt (Auswirkung minimieren)

2.5.4 Mangelhafte Professionalität der Projektmitarbeiter

Mangelhafte Codequalität und oder fehlerhafter Einsatz von Entwicklungswerkzeuge wie IDE und Git. Unprofessionelles Verhalten der Projektmitarbeiter.

Massnahmen zur Risikominderung

- Regelmässige Code-Review welche selbstständig zu organisieren sind (Eintrittswahrscheinlichkeit minimieren)
- Befolgen der Clean Code Leitlinien und automatisierte Überprüfung der Qualität mittels Jenkins (Eintrittswahrscheinlichkeit minimieren)
- Für die Entwicklungswerkzeuge sollen jeweils im README die wichtigsten Befehle festgehalten werden, damit alle gleich mit den Werkzeugen arbeiten. (Eintrittswahrscheinlichkeit minimieren)
- Selbstständiges Einarbeiten in IDE und Git (Eintrittswahrscheinlichkeit minimieren)
- Regelmässige Datensicherung mit TimeMachine oder ähnlichen Dienstprogrammen (Auswirkung minimieren)
- Einsatz aktueller Virensoftware und regelmässiges aktualisieren der Datenbanken auf allen Arbeitsgeräten (Eintrittswahrscheinlichkeit minimieren)

2.5.5 COVID-19

Der Coronavirus könnte sich jederzeit auch in Europa und somit in der Schweiz verbreiten. Dies könnte unter Umständen, zu einer Schliessung der Schulen führen, wie dies bereits in Asien geschehen ist.

Massnahmen zur Risikominderung

- Im Falle einer Schliessung der Schule, wird das Team weiterhin eng zusammenarbeiten (Eintrittswahrscheinlichkeit minimieren)
- Verstärkter Nutzen der gängigen Kommunikationstools wie Skype oder WhatsApp (Eintrittswahrscheinlichkeit minimieren)

3 Projektunterstützung

3.1 Tools für Entwicklung, Test und Abnahme

Im Projektmanagement setzen wir folgende Tool ein: In der Entwicklung setzen wir folgende Tool ein:

Bereich	Tool
Vorgehensmodell: SoDa	GitLab
Datenaustausch	GitLab, OneDrive
Kommunikation	WhatsApp, Skype, Zoom
Dokumentation	Microsoft Office, Draw.io, yFiles

Bereich	Tool
Entwicklungsumgebung	IntelliJ IDEA, Eclipse, VS Code
Programmiersprache	Java
Versionskontrolle	GitLab
Testing	JUnit, Integrations- und Systemtests
Build	Maven
Continuos Integration	Jenkins

3.2 Konfigurationsmanagement

Das Konfigurationsmanagement soll die Einhaltung von Regeln für einen organisatorischen und verhaltensmässigen Lebenslauf eines Produkts und seiner Configuration Items (Konfigurationseinheiten) gewährleisten. Ein Configuration Item ist eine beliebige Kombination aus Hardware, Software und / oder Dienstleistung. In diesem Projekt sind das die Dokumentationen, Komponenten und Interfaces. Im Kapitel Releasemanagement sind eben erwähnte ausführlich aufgelistet.

3.3 Releasemanagement

Das Releasemanagement befasst sich mit der Planung und Durchführung der Veröffentlichung. Dieses Projekt beinhaltet zwei Releases:

- **Release 1:**Zwischenabgabe (SW10)
- **Release 2:**Schlussabgabe (SW14)

Configuration Item	Release 1	Release 2
Projektmanagementplan	1.0.0	1.0.2
Systemspezifikation	1.0.0	1.0.2
Testplan	1.0.0	1.1.0
GameOfLife	1.0.0-SNAPSHOT	1.0.0-SNAPSHOT
Logger	1.0.0	1.0.0
LoggerSetup	1.0.0	1.0.0
LoggerCommon	1.0.0-SNAPSHOT	1.0.0-SNAPSHOT
LoggerComponent	1.0.0-SNAPSHOT	1.0.0-SNAPSHOT
LoggerServer	1.0.0-SNAPSHOT	1.0.0-SNAPSHOT
LoggerClient	-	1.0.0-SNAPSHOT
StringPersistor	5.0.3	5.0.3
StringPersistorFile	1.0.0-SNAPSHOT	1.0.0-SNAPSHOT

4 Teststrategie und -Drehbuch

Teststrategie und Drehbuch sind in einem seperaten Dokument zu finden.

5 Anhang

5.1 Product Backlog

Das Product Backlog beinhaltet alle Anforderungen des Kunden. Diese werden laufend zusammen mit dem Product Owner ausgearbeitet und priorisiert. Vollständiger Backlog ist im GitLab ersichtlich.

Issue ID	Titel	Sprint	Label
1	Interface Definieren und Ausarbeiten	Initialisierungsphase	
2	Projektdokumentation vorbereiten	Initialisierungsphase	
3	Milesstones in Gitlab erfassen	Initialisierungsphase	
4	ProduktBacklog Issues erfassen	Initialisierungsphase	
5	Risikoanalyse	Initialisierungsphase	
7	Log Statements implementieren	Sprint 1	Game-of-Live
8	Logger Interface implementieren	Sprint 1	Game-of-Live
9	Logger Interface definieren	Sprint 1	Logger
10	Logger Klassen implementieren	Sprint 1	Logger
11	TCP/IP Kommunikation zu Log-Server implementieren	Sprint 2	Logger
12	Log-Level Filter implementieren	Sprint 2	Logger
13	Log Buffer implementieren	Sprint 2	Logger
14	TCP/IP Kommunikation implementieren	Sprint 2	Log Server
15	StringPersistorFile Interface implementieren	Sprint 2	Log Server
16	Server Logger Klassen implementieren	Sprint 2	Log Server
17	I/O Klassen implementieren	Sprint 2	Log Server
18	Eigener Logger implementieren	Sprint 1	Log Server
19	Lösung für I/O Bottleneck beim Log-File beschreiben lösen	Sprint 2	Log Server
20	Log Message definieren und implementieren	Sprint 1	Logger
21	API für Logger auf Client		Logger
27	Testplan dokumentieren	Sprint 2	Documentation
28	Systemspezifikation erstellen	Sprint 2	Documentation

5.2 Sprintplanung Sprint 1

Beinhaltet die Planung für den ersten Sprint. Das Issue 'Log Statements implementieren' (Issue ID: 6) wurde in vier Issues aufgebrochen, weil jedes Teammitglied ein paar Klassen abgedeckt bei der Implementierung. Auf eine separate Aufführung dieser wird hier verzichtet, jedoch sind die Issues im GitLab ersichtlich.

Issue ID	Titel	Sprint	Label
6	Log Statements implementieren	Sprint 1	Game-of-Live
7	Logger Interface implementieren	Sprint 1	Game-of-Live
8	Logger Interface definieren	Sprint 1	Logger
9	Logger Klassen implementieren	Sprint 1	Logger
17	Eigener Logger implementieren	Sprint 1	Log Server
20	Log Message definieren und implementieren	Sprint 1	Logger

5.2.1 Sprintreview

Im Sprint 1 konnten alle geplanten Issues abgearbeitet werden mit der Ausnahme von 'Logger Interface implementieren' (Issue ID: 7). Aufgrund von Unklarheiten konnte dies noch nicht endgültig erledigt werden. Das Issue wird somit in den Sprint 2 genommen. Bei den erfolgreich bearbeiteten Issues konnten die Vorgaben bezüglich des Zeitaufwandes eingehalten werden. Es wurden, wo sinnvoll Tests implementiert. Mehr Informationen zum Testing können im Hauptteil unserer Arbeit gefunden werden.

Issue ID	Titel	Sprint	Label
7	Logger Interface implementieren	Sprint 1	Logger

5.2.2 Retrospektive

Grundsätzlich hat vieles gut funktioniert. Jedoch ist schnell aufgefallen, dass manchmal mehrere Issues für dieselbe Aufgabe gemacht wurden, was zu Unsicherheiten im Team geführt hat. Dies soll zukünftig vermieden werden. Bei einem weiteren Projekt sollte mehr Arbeit bereits in den Sprint 1 genommen werden, um mehr Puffer zu haben, denn Sprint 2 ist merklich grösser vom Workload her als der erste Sprint.

5.3 Sprintplanung Sprint 2

Beinhaltet die Planung für den zweiten Sprint.

Issue ID	Titel	Sprint	Label
7	Logger Interface implementieren	Sprint 2	Game-of-Live
11	TCP/IP Kommunikation zu Log-Server implementieren	Sprint 2	Logger
12	Log-Level Filter implementieren	Sprint 2	Logger
13	Log Buffer implementieren	Sprint 2	Logger
14	TCP/IP Kommunikation implementieren	Sprint 2	Log Server
15	StringPersistorFile Interface implementieren	Sprint 2	Log Server
16	Server Logger Klassen implementieren	Sprint 2	Log Server
17	I/O Klassen implementieren	Sprint 2	Log Server
19	Lösung für I/O Bottleneck beim Log-File beschreiben, lösen	Sprint 2	Log Server
27	Testplan dokumentieren	Sprint 2	Documentation
28	Systemspezifikation erstellen	Sprint 2	Documentation

5.3.1 Sprintreview

Im Sprint 2 konnten alle geplanten Issues bearbeitet werden. Einzige Ausnahme ist das Issue 19. Dieses wird im nächsten Sprint bearbeitet werden. Mehrheitlich konnte der geplante Zeitrahmen eingehalten werden. Bei Problemen hat jeweils das ganze Team geholfen, diese zu beheben.

Issue ID	Titel	Sprint	Label
19	Lösung für I/O Bottleneck beim Log-File beschreiben, lösen	Sprint 2	Log Server

5.3.2 Retrospektive

Von der Planung an sich her war der zweite Sprint besser geplant, wobei immer noch Verbesserungspotenzial besteht. Wie bereits am Ende des ersten Sprints angemerkt, wurde der zweite Sprint viel grösser, was logischerweise zu einem erhöhten Aufwand führte. Dieser Fall soll zukünftig um jeden Preis verhindert werden, da der Puffer am Schluss fehlt. Die Teamarbeit an sich hat aber weiterhin vorbildlich funktioniert.

5.4 Sprintplanung Sprint 3

Beinhaltet die Planung für den dritten Sprint.

Issue ID	Titel	Sprint	Label
19	Lösung für I/O Bottleneck beim Log-File beschreiben lösen	Sprint 3	Log Server
31	Optimierung lokaler Logger	Sprint 3	
32	Unterscheidung von Logs anhand Application ID	Sprint 3	Log-Daten
33	Anpassungen im PMP gemäss Rückmeldungen M. Bättig	Sprint 3	
35	JavaDoc in StringPersitorFile ergänzen	Sprint 3	
36	Logs verfeinern im Game	Sprint 3	
37	Performance Stringpersistor	Sprint 3	
38	Classloader load jar of logger implementation	Sprint 3	
41	Strategy Pattern - Log in verschiedenen Formaten speichern	Sprint 3	

5.4.1 Sprintreview

Im Sprint 3 konnten alle geplanten Issues bearbeitet werden. Einzige Ausnahme ist das Issue 38. Dieses wird im nächsten Sprint bearbeitet werden. Mehrheitlich konnte der geplante Zeitrahmen eingehalten werden. Bei Problemen hat jeweils das ganze Team geholfen, diese zu beheben.

Issue ID	Titel	Sprint	Label
38	Classloader load Jar of logger implementation	Sprint 3	

5.4.2 Retrospektive

Von der Planung her war der dritte Sprint besser geplant, wobei immer noch Verbesserungspotenzial besteht. Lediglich eine Issue konnte nicht beendet werden, war aber auch dem Auftraggeber geschuldet. Nötige Informationen zum Jar-Loader konnten nicht rechtzeitig eingeholt werden (Vorlesung zu diesem Thema fand erst nach diesem Sprint statt).

5.5 Sprintplanung Sprint 4

Beinhaltet die Planung für den vierten Sprint.

Issue ID	Titel	Sprint	Label
38	Classloader load Jar of logger implementation	Sprint 4	Logger
40	RMI Implementation für Viewer	Sprint 4	Log Server
42	Server runnable from Jar	Sprint 4	Logger
43	Dokumentation Entscheid Logische Uhren	Sprint 4	Dok

5.5.1 Sprintreview

Im Sprint 4 konnten alle geplanten Issues erfolgreich bearbeitet werden.

5.5.2 Retrospektive

Es wurde gelernt aus den vergangenen Sprints und dadurch konnte dieser Sprint erfolgreich durchgeführt werden.

5.6 Meilensteinberichte

5.6.1 Meilensteinbericht 1

Termin Meilenstein 1 Der erste Meilenstein findet zu Beginn der dritten Semesterwoche statt.

Beschreibung Meilenstein 1 Beim ersten Meilenstein im Projekt geht es um die allgemeine Besprechung sowie Freigabe der Interfaces, welche von den Interface-Komitees vorgestellt werden. Die Entscheidung für ein Interface-Set wird im Plenum gefällt. Delegierter unseres Teams wird Maurizio Hostettler sein. Neben der Präsentation soll die Organisation im Team koordiniert werden.

Meilensteinziele/-vorgaben

- 1. Präsentation der Interfaces der drei Interface-Komitees
- 2. Diskussion der Interfaces im Plenum
- 3. Abstimmung im Plenum über das zu implementieren Interface - Freigabe
- 4. PMP Dokumentation starten (SoDa-Rollen, Risikoliste, Product Backlog, Sprintplanung 1)

Meilensteinzielerreichung

- 1. Interfaces wurden präsentiert und freigegeben
- 2. PMP Dokumentation gestartet
  - a. SoDa-Rollen sind definiert
  - b. Risikoliste ist erstellt
  - c. Product Backlog ist definiert
  - d. Die erste Sprintplanung steht

5.6.2 Meilensteinbericht 2

Termin Meilenstein 2 Die zweite Meilenstein findet zu Beginn der achten Semesterwoche statt.

Beschreibung Meilenstein 2 Beim zweiten Meilenstein geht es um den ersten Release des Projektes. Es muss eine Demo des Message Logging Systems vorgeführt werden. Die Programme des Systems müssen ausserhalb der IDE lauffähig sein, in den vorgegebenen Projekten auf GitLab verwaltet werden, auf dem Buildserver laufend integriert worden sein, automatisierte Tests ohne Fehler durchlaufen können und eine begründete minimale Codeabdeckung erfüllen. Zudem muss eine vollständige Dokumentation abgeliefert werden.

Anforderungen an das Message Logging System

Nr.	Bezeichnung
1	Die Logger-Komponente muss als Komponente (mit Interfaces) realisiert werden.
2	Bei jedem Log-Eintrag hat die aufrufende Applikation einen **Message-Level** mitzugeben. Über die API der Logger-Komponente kann ein Level-Filter gesetzt werden. Damit kann definiert werden, welche Meldungen (mit welchem Level) tatsächlich übertragen werden. Dieser Level kann zur Laufzeit geändert werden.
3	Die Logger-Komponente benötigt folgende Software-Interfaces: **Logger**: Message erzeugen und eintragen. Eine Applikation kann via Methodenaufruf Messages (Textstrings) loggen. **LoggerSetup**: Dient zur Konfiguration des Message Loggers. Weitere Schnittstellen sind wo sinnvoll individuell zu definieren.
4	Die Log-Ereignisse werden durch Logger-Komponente und den Logger-Server kausal und verlässlich aufgezeichnet.
5	Die Logger-Komponente ist austauschbar und plattformunabhängig zu realisieren. Der Komponentenaustausch muss ausserhalb der Entwicklungsumgebung und **ohne** Code-Anpassung, d.h. ohne Neukompilation möglich sein.
6	Es muss möglich sein, dass mehrere Instanzen der Logger-Komponente parallel auf den zentralen Logger-Server loggen.
7	Die dauerhafte Speicherung der Messages erfolgt auf dem Server in einem einfachen, lesbaren Textfile. Das Textfile enthält mindestens die Quelle der Logmeldung, zwei Zeitstempel (Erstellung Message, Eingang Server), den Message-Level und den Message-Text.
8	Für das Schreiben des Textfiles auf dem Server ist die vorgegebene Schnittstelle **StringPersistor** zu verwenden und auch dafür eine passende Komponente (StringPersistorFile) zu implementieren.
9	Verwenden Sie Adapter (GoF-Pattern) um Daten in strukturierter Form in den Payload Parameter der **StringPersistor** Schnittstelle übergeben zu können. Testen Sie die Adapter mittels Unit Tests. Die vorgegebene Schnittstelle **StringPersistor** muss eingehalten werden.

Meilensteinziele

- 1. Ein Logger-Interface zur Verfügung stellen
- 2. Logger-Komponente implementieren
- 3. Logger-Server implementieren
- 4. StringPersistor-Komponente implementieren
- 5. Events innerhalb von Game-of-Life über das Logger-Interface mit der Logger-Komponente auf dem Logger-Server mittels StringPersistor loggen
- 6. Vollständige Dokumentation

Meilensteinzielerreichung

- 1. Ein Logger-Interface steht zur Verfügung
- 2. Logger-Komponente wurde implementiert
- 3. Logger-Server wurde implementiert
- 4. StringPersistor inkl. StringPersistorFile wurde implementiert
- 5. Events innerhalb von Game-of-Life werden auf dem Logger-Server geloggt
- 6. Die Dokumentation ist vollständig

5.6.3 Meilensteinbericht 3

Termin Meilenstein 3 Die dritte Meilenstein findet zu Beginn der vierzehnten Semesterwoche statt.

Beschreibung Meilenstein 3 Beim dritten Meilenstein geht es um den zweiten und finalen Release des Projektes. Es muss eine Demo des Message Logging Systems vorgeführt werden. Die Programme des Systems müssen ausserhalb der IDE lauffähig sein, in den vorgegebenen Projekten auf GitLab verwaltet werden, auf dem Buildserver laufend integriert worden sein, automatisierte Tests ohne Fehler durchlaufen können und eine begründete minimale Codeabdeckung erfüllen. Zudem muss eine vollständige Dokumentation abgeliefert werden. Zudem müssen Komponenten zwischen den Teams ausgetauscht werden können.

Anforderungen an das Message Logging System

Nr.	Bezeichnung
10	Die **Qualitätsmerkmale** der vorgegebenen Schnittstelle **StringPersistor** werden erreicht.
11	Das Speicherformat für das Textfile (siehe Feature 7) soll über verschiedene, austauschbare Strategien (GoF-Pattern) leicht angepasst werden können. Testen Sie die Strategien mittels Unit Tests.
12	**Statische Konfigurationsdaten** der Komponenten (z.B.: Informationen bezüglich Erreichbarkeit des Servers) sind zu definieren und müssen **ohne Programmierung anpassbar** sein.

13	Die Socket Verbindung zwischen Logger-Komponente und Logger-Server muss so robust implementiert werden, dass diese mit Netzwerkunterbrüchen umgehen kann. Für die dazu notwendige Zwischenspeicherung der Messages soll die bereits für den Logger-Server entwickelte **StringPersistorFile** Komponente wiederverwendet werden.
14	Implementation eines Viewers, welcher sich per **RMI** mit dem Logger-Server verbindet und alle eintreffenden Messages online anzeigt, funktioniert nach dem **Push-Prinzip**. Es sollen mehrere Instanzen des Viewers auf den Logger-Server zugreifen können. Die Anzeige umfasst zwei Zeitstempel, einen mit der Eingangszeit beim LoggerServer und derjenige der Logmessage. Beim LoggerViewer handelt es sich um eine Applikation mit einer graphischen Benutzeroberfläche. (Swing oder JavaFX, keine Webtechnologie).

Meilensteinziele

- 1. StringPersistor soll Qualitätsmerkmale erfüllen
- 2. Strategy-Pattern implementieren
- 3. Client Lokaler-Logger implementieren
- 4. RMI implementieren

Meilensteinzielerreichung

- 1. StringPersistor erfüllt Qualitätsmerkmal
- 2. Strategy-Pattern wurde implementiert
- 3. Client Lokaler-Logger wurde implementiert
- 4. RMI wurde implementiert