Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Team 5

Fischer Frederico Küng Matthias Werlen Oliver Majkel Zivkovski

Message Logger

Projektmanagement-Plan

Version 3.0.0

1. Projektorganisation

1.1. Organisationsplan, Rollen & Zuständigkeiten

Der VSK-Projektauftrag «Message Logger» wird von einer 4-köpfigen Gruppe durchgeführt. Im Rahmen dieses Auftrages soll ein alternatives und komponentenbasierendes Message Logging System entwickelt werden, welches unabhängig von der Plattform und auch auf mehreren Hosts verteilt verwendet werden kann.

Als Projekt- und Vorgehensmodell wird SoDa (Software Development agile) verwendet, welches hauptsächlich 4 Rollen kennt. Nachfolgend werden die Rollen und Zuständigkeiten im Rahmen dieses Projektes näher vorgestellt.

Majkel Zivkovski - Projektleiter:

Majkel Zivkovski 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. Dieser Rahmenplan gibt eine grobe Vorstellung des übergeordneten Projektablaufs mit den wichtigsten Meilensteinen und Terminen.

Frederico Fischer - Product Owner / Delegierter des Interface-Komitees:

Frederico Fischer wird in der Konzeptions- und Realisierungsphase eine entscheide Stellung 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 Delegierter des Interface-Komitees agieren.

Oliver Werlen – Scrum Master:

Oliver Werlen wird im Projekt die Rolle des Scrum Master einehmen. In dieser Rolle wird er als Vermittler zwischen dem Scrum Team und dem Product Owner agieren. Zusätzlich unterstützt er ein korrektes Projektvorgehen und stellt die Qualität der Leistung sicher.

Matthias Küng, Oliver Werlen, Majkel Zivkovski – Scrum Team:

Matthias Küng nimmt aufgrund seiner langjährigen Entwicklererfahrung eine wichtige Funktion innerhalb des Scrum Teams ein. Er wird sicherstellen, dass der Code den Anforderungen entspricht und wird zusätzlich den anderen Teammitgliedern bei der Entwicklung unter die Hände greifen.

1.2. Projektstrukturplan

Anhand des Projektstrukturplanes soll das Projekt in plan- und kontrollierbare Elemente auf eine grobe Art gegliedert werden. Im Rahmen dieses Projektes wurden 4 Phasen definiert, welche im Folgenden näher beschrieben werden. Der grafische Projektstrukturplan ist im Anhang unter 6.1. beigefügt.

1.2.1 Vorprojektphase

Das Hauptziel der Vorprojektphase ist es sich als Team zu finden und zu organisieren sowie eine agile Rollenverteilung des Teams vorzunehmen. Ein weiteres Ziel umfasst die Codeanalyse der Simulation «Game-of-Life». Zudem wird bereits hier die Dokumentation gestartet und fortlaufend erweitert.

1.2.2 Initialisierungsphase

In der Initialisierungsphase wird das Interface Komitee je ein Set von Interfaces für die Logger-Komponente definieren und dokumentieren. Im Plenum wird dann der Entscheid für ein Interfaces getroffen. Zudem definiert der Product Owner Frederico Fischer den Product Backlog. Beim Product Backlog handelt es sich um eine Liste mit den priorisierten Anforderungen des Projektes (zu finden im Anhang 6.3.). Der Projektleiter Majkel Zivkovski wird in dieser Phase einen Rahmenplan mit den wichtigsten Meilensteinen definieren.

1.2.3 Konzeptionsphase

In der Konzeptionsphase beginnt die Entwicklung des alternativen und komponentenbasierten Message Logging Systems. Aufbauend aus den priorisierten Anforderungen des Product Backlogs und des Rahmenplans wird Frederico Fischer in Zusammenarbeit mit dem Team eine rollende Detail-Planung (Sprintplanung) ausarbeiten. Weitere Tätigkeiten dieser Phase umfasst der Testplan, die

Entwicklung sowie Testdurchführung der Komponenten. Des Weiteren findet zu Beginn dieser Phase das Risikomanagment statt.

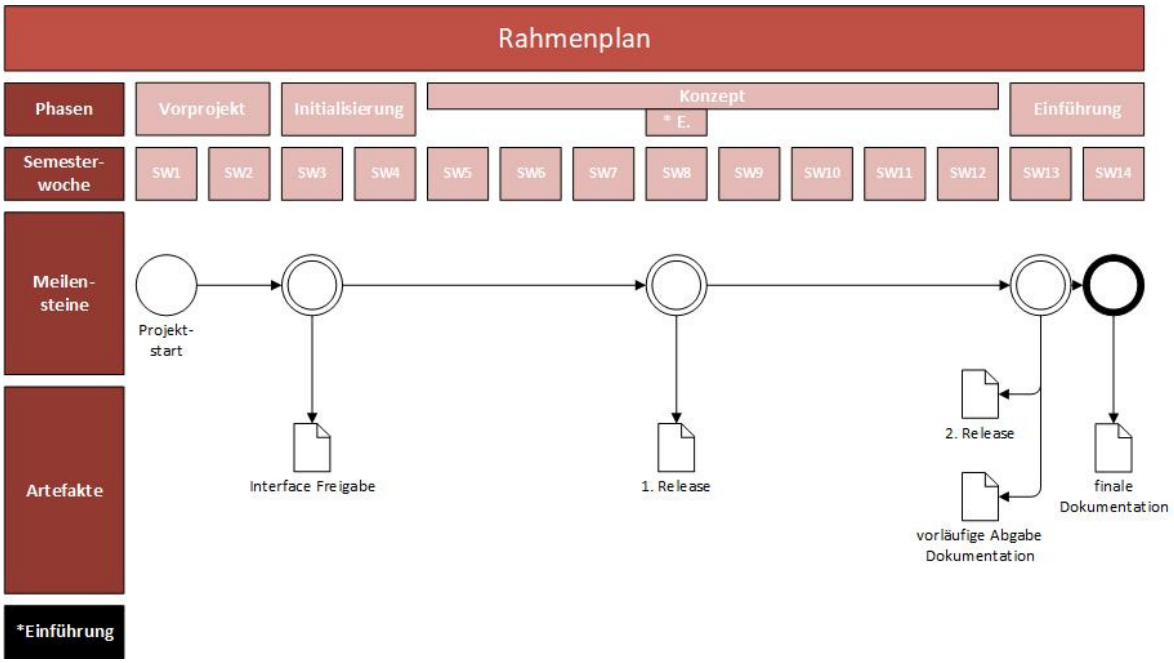
1.2.4 Einführungsphase

Die Einführungsphase umfasst die zwei Releases in Semesterwoche 8 und die finale Abgabe in Semesterwoche 13.

2. Projektführung

2.1. Rahmenplan

Der Rahmenplan umfasst ein grobes Ablaufschema innerhalb des VSK-Projektes und soll dem Team eine grobe Vorstellung des übergeordneten Projektablaufs mit den wichtigsten Meilensteinen und Terminen geben. Dieser wird nachfolgend aufgeführt und umfasst Semesterwoche mit dem entsprechenden Termin / Meilenstein:



Semesterwoche 1:

- Organisation des Teams.

Semesterwoche 2:

- Start Dokumentation
- Codeanalyse
- agile Rollenverteilungen gemäss SoDa Projektmanagement definieren

Semesterwoche 3:

- Interface Besprechung & Freigabe (Meilenstein)
- Erstellung Product Backlog
- Fertige Rahmenplanung

- Sprintplanung 1
- Risikomanagement

Semesterwoche 4:

- Start Sprint 1
- Erstellung Unit Tests für Sprint 1
- Start Entwicklung der Komponenten in Sprint 1

Semesterwoche 5:

- Testdurchführung (Unit Tests) der Komponenten in Sprint 1
- Ende Sprint 1

Semesterwoche 6:

- Start Sprint 2
- Erstellung Unit Tests für Sprint 2
- Start Entwicklung der Komponenten in Sprint 2

Semesterwoche 7:

- Testdurchführung (Unit Tests) der Komponenten in Sprint 2
- Integrationstests der Schnittstellen und fertiggestellten Systemkomponenten
- Ende Sprint 2

Semesterwoche 8:

- 1. Release Zwischenabgabe (Meilenstein)

Semesterwoche 9:

- Start Sprint 3
- Erstellung Unit Tests für Sprint 3
- Start Entwicklung der Komponenten in Sprint 3

Semesterwoche 10:

- Testdurchführung (Unit Tests) der Komponenten in Sprint 3
- Integrationstests der Schnittstellen und fertiggestellten Systemkomponenten
- Ende Sprint 3

Semesterwoche 11:

- Start Sprint 4
- Erstellung Unit Tests für Sprint 4
- Start Entwicklung der Komponenten in Sprint 4

Semesterwoche 12:

- Ende Sprint 4
- Testdurchführung (Unit Tests) der Komponenten in Sprint 4
- Integrationstests der Schnittstellen und fertiggestellten Systemkomponenten

Semesterwoche 13:

- 2. Release Demonstration und Präsentation des Message Logging Systems (Meilenstein)
- Vorläufige Abgabe der Dokumentation (Meilenstein)

Semesterwoche 14:

- Abgabe der endgültigen Dokumentation (Meilenstein)

Die ausgerollte Detailplanung mit den einzelnen Sprints wird im Anhang aufgeführt.

2.2. Projektkontrolle

Im Rahmen der Projektkontrolle werden folgende Tools eingesetzt, um einen möglichst reibungslose Projektführung zu ermöglichen:

- Product Backlog
- Formulierung von Definition of Done der Tasks
- Sprintplanungen
- Soll/IST Zeitvergleiche der einzelnen Issues
- Sprintreviews
- Meilensteinberichte

2.3. Aufwandschätzung

Die Aufwandschätzung des Projektes findet innerhalb der einzelnen Issues statt. Jedes Teammitglied soll bei der Erstellung eines neuen Issues eine ungefähre Fertigstellungszeit angeben. Bei Beendigung des Tasks wird die effektive aufgewendet Zeit eingetragen, um so einen Soll/Ist-Vergleich zu ermöglichen und daraus Lesson's Learned zu generieren. Die Lesson's Learned sollen in der nächsten Sprintplanung für bessere Schätzungen sorgen. Die aktuellen Soll/Ist Vergleiche sind im CSV_File.2.0.0 aufgeführt.

2.4. Risikomanagement

Das Risikomanagement sollte, wie in jedem Projekt, in einem frühen Stadium beginnen. Im Rahmen dieses Projektes haben wir uns in Semesterwoche 3 zusammengesetzt und im Plenum eine Risikomatrix (siehe Anhang 5.2.) erstellt. Diese stellt die Wahrscheinlichkeit des Auftretens eines unerwünschten Ereignisses in Bezug zu dessen Schadensausmass.

Anhand der Matrix haben wir 4 Risikoquellen identifizieren, welche im Rahmen des VSK Projektes kritisch werden könnten. Um das Risiko für diese 4 Quellen zu reduzieren haben wir geeignete Massnahmen getroffen, welche nachfolgend aufgeführt werden:

Risikoquelle 1 – schlechtes Design

Reduktion der Eintrittswahrscheinlichkeit

- wöchentliche Teamtreffen mit dem Ziel den Code gemeinsam zu überarbeiten und dadurch einen Lernprozess zu generieren

Reduktion des Schadensausmass

- kontinuierliches Refactoring des Source Codes
- monatliche Überprüfung und Verbesserung der erfahreneren Software Entwickler im Team

Risikoquelle 2 – falscher Umgang mit Git

Reduktion der Eintrittswahrscheinlichkeit

- Einlesen in die Thematik
- Die Funktionsweisen von Git zunächst lokal erlernen

Reduktion des Schadensausmass

- Eigenverantwortung der Teammitglieder

- lokale Sicherungen erstellen

Risikoquelle 3 – Ressourcenmangel an Mitarbeitern / Zeitknappheit

Reduktion Eintrittswahrscheinlichkeit

- Deadlines definieren
- Kontrolle der Deadlines
- fixe Anzahl Arbeitsstunden pro Woche vereinbaren
- strikte Rollenverteilung

Reduktion Schadensausmass

- Priorisierung der Tätigkeiten
- Puffer in Deadlines einbauen für Fehlerbehebung

Risikoquelle 4 – Fehler im Projektmanagement

Reduktion Eintrittswahrscheinlichkeit

- geeignete Wahl bei Rollenverteilung (Majkel Zivkovski hat bereits ein Studium im General Management und praktische Erfahrung im Projekt Management)
- Majkel Zivkovski studiert seine alten Studiumsunterlagen im Bereich Projektmanagement
- Aufgaben sollten klar formuliert werden

Reduktion Schadensausmass

- regelmässige Reviews um Fehler im Plenum schnell zu identifizieren und zu reagieren
- Bei Fehler sollten Schlüsse gezogen und Lösungen gesucht werden, damit diese nicht zu einem späteren Zeitpunkt erneut auftreten

Für den zweiten Release wurde eine weitere Risikoquelle identifiziert, an welche im ersten Release nicht gedacht wurde und sich deshalb auch nicht in der Risikomatrix befindet. Da der Winter vor der Tür steht, häufen sich auch krankheitsbedingte Ausfälle. Das Team hat hierzu einige Massnahmen definiert, die nachfolgend aufgeführt werden:

Risikoquelle 5 – krankheitsbedingte Ausfälle

Reduktion Eintrittswahrscheinlichkeit

- gesunde Ernährung
- warme Bekleidung während der nächsten zwei Sprints

- kranke Mitmenschen bestmöglich meiden

Reduktion Schadensausmass

- stetiger Wissensaustausch unter den Teammitglieder
- aktuelle Arbeitsstände schnellstmöglich pushen
- bei der Verteilung der Aufgaben Ersatzperson bei Ausfall bestimmen

3. Projektunterstützung

3.1. Tools für Entwicklung, Test & Abnahme

Für das Projektmanagement sind folgende Tools im Einsatz:

Bereich	Tool
Vorgehensmodell: SoDa	GitLab
Datenaustausch	GitLab, OneDrive
Dokumentation	Microsoft Office, UMLet, Visio 2016

Für die Entwicklung sind folgende Tools im Einsatz:

Bereich	Tool
Entwicklungsumgebung	IntelliJ IDEA, NetBeans, Eclipse
Programmiersprache	Java 1.8
Versionskontrolle	GitLab, SmartGit, Sourcetree
Testing	JUnit, Integrationstests, Systemtests
Build	Maven 3.6.0
Continuous Integration	Jenkins 2.195

3.2. Konfigurationsmanagement

Das Konfigurationsmanagement ist eine Managementdisziplin, die organisatorische und verhaltensmäßige Regeln auf den Lebenslauf eines Produkts und seiner Konfigurationseinheiten (Configuration Items) von Entwicklung über Herstellung und Betreuung bis hin zur Entsorgung anwendet (Wikipedia, 2019).

Gemäss Wikipedia versteht man unter einer Konfigurationseinheit sämtliche an führenden Geschäftsprozessen beteiligten Betriebsmittel. Beispiele wären Dokumentationen, Software, Server und diverse Weitere (Wikipedia, 2016). Im Rahmen des VSK Projektes stellen die Dokumentation, die Komponenten sowie die Interfaces die Konfigurationseinheiten dar. Eine genau Auflistung der projektrelevanten Konfigurationseinheiten finden sie im Kapitel 3.3 Releasemanagement.

3.3. Releasemanagement

Releasemanagement bedeutet die Planung und Durchführung der Veröffentlichung, von der Idee bzw. den ersten Anforderungen bis zum Erreichen des Endbenutzers (Wikipedia, 2019).

Das Logger-Projekt umfasst zwei Releasezyklen (Zwischenabgabe SW8 und Schlussabgabge SW13), welche nachfolgend mit den einzelnen Konfigurationseinheiten aufgeführt werden.

Konfigurationseinheit	Release 1	Release 2
Projektmanagementplan	1.0.0	3.0.0
Testplan	1.0.0	2.0.0
CSV-Export GitLab	1.0.0	2.0.0
Systemspezifikation	1.0.0	3.0.0
Logger(API)	1.0.0	1.0.0
LoggerSetup(API)	1.0.0	1.0.0
LoggerCommon	1.0.0	2.0.0
LoggerComponent	1.0.0	2.0.0
LoggerServer	1.0.0	2.0.0
LoggerViewer		1.0.0
StringPersistor(API)	5.0.1	5.0.1
StringPersistor	1.0.0	1.4.0
LogPersistor(API)	1.0.0	1.0.0
Game-Of-Life	1.0.0	1.0.0
StringPersistorFileStrategy(API)		1.0.0
RemoteViewHandler(API)		1.0.0
RMIRegistration(API)		1.0.0

4. Testplan

4.1. Teststrategie

Unsere Teststrategie beruht grössten Teils auf Integration Tests. Jedoch werden auch Fakes verwendet, um Unit-Tests zu nutzen. Des Weiteren wurden vor der Schlussabgabe die Muss-Features von Teammitgliedern getestet. Hierzu wurden vorgängig Testfälle und deren erwartete Resultate definiert. Anschliessend wurden diese durchgeführt und deren Resultate dokumentiert.

4.2. Feature Tests

Auf Seite 4 des Projektauftrags sind alle Muss-Features der Logger Componente aufgelistet. In den folgenden Punkten wird erläutert, wie diese getestet wurden.

4.2.1. Logger als Component

Der Logger muss als Komponte mit Interfaces realisiert werden.
LoggerComponent muss das Interface LoggerSetup implementieren.
Der Logger muss während des Spielbetriebs ausgetauscht werden können. Dazu werden die get-Methoden aus dem Interface aufgerufen. Die Daten des neuen Loggers werden aus einem File eingelesen. Die Funktionalität beweist, dass LoggerComponent keine Abhängigkeiten zu anderen Projekten oder Componenten hat.

4.2.2. MessageLevel setzen

Das gewünschte Message Level wird durch die dazugehörige setter-Methode von LoggerSetup gesetzt. Dies ermöglicht eine Abänderung während der Laufzeit. Es werden dann nur Message mit diesem Level geloggt. Alle anderen werden ignoriert. Die dazugehörige Überprüfung wird beim Erstellen der LogMessage getätigt

Das Testen des LogLevels wird als UnitTest realisiert. Mittels der Methode testLogLevelCorrect(), testLogLevelFalse() werden jeweils logs mit dem richtigen oder mit dem falschen LogLevel ausgelöst. Anschliessend wird ein Assert auf die lokale List gemacht, da diese als lokale Kopie zum Server dient. In dieser dürfen nur LogMessages mit dem richtigen LogLevel sein.

4.2.3. Interfaces Logger, LoggerSetup

LoggerComponent muss das Interface Logger implementieren, die Klasse LoggerComponentSetup das Interface LoggerSetup.
Das Interface LoggerSetup stellt eine Factory zur Verfügung. Diese kann mittels Unit Test testLoggerSetupFactory in unserem Projekt getestet werden.

Da die Komponenten untereinander austauschbar sein müssen und dies getestet wird, wird die Funktionalität des Interfaces bereits hier getestet. Würde dies nicht korrekt implementiert werden, wären diese nicht austauschbar.

In einem Testfall wird ein LoggerServer erstellt. Der Client (Logger) wird mit den korrekten Parametern erstellt und verbindet sich auf den Server. Anschliessend wird eine logMessage erstellt und an den Server gesendet. Mittels Assert wird das angekommene mit dem gesendeten verglichen.

4.2.4. Zuverlässiges und kausales Aufzeichnen von Logs

Um dies zu gewährleisten, werden in der Methode testLogMethod() 10 logs ausgelöst.
Es wird anschliessend die Anzahl der in der lokalen List gespeicherten Elemente mit Assert auf 10 getestet, anschliessend wird dasselbe auf dem DemoServer überprüft.

Das selbe wird anschliessend mit der Methode testLogMethodExtreme() getestet. Hier werden 1000 logs ausgelöst. Dies ist ein Extremtest, somit kann aber gewährleistet werden, dass der Server sowie der Logger auch bei vielen Anfragen zuverlässigt loggt und die Messages korrekt zwischengespeichert bzw. abgespeichert werden.

4.2.5. Austauschbarkeit der LoggerComponente ausserhalb der IDE

Die Factory vom LoggerSetup benötigt 2 Parameter, um ein LoggerSetup zu erstellen. Beim ersten handelt es sich um den Pfad zum .jar des jeweiligen LoggerComponent. Des Weiteren ist der Projektpfad zur Klasse LoggerComponent anzugeben. Die Werte werden über ein Config-File in das Programm eingelesen und bei einer Änderung wird ein Autoreload im GameOfLife ausgeführt, sodass direkt und ohne Neukompilierung mit dem neuen Logger geloggt werden. Ausserdem werden

in diesem Config-File auch die Werte für die Setter-Methoden von LoggerSetup gespeichert. Das Einlesen funktioniert identisch.

Diese Funktionen werden mittels Unit-Tests getestet. Die testLoadFromFile() vergleicht die eingelesenen Werte mittels Assert auf deren Richtigkeit. Die einzelnen setter-Methoden werden im Komponenten LoggerComponent mittels Unit-Tests getestet. Das Vorgehen dabei ist dasselbe.

4.2.6. Mehrere Instanzen lokal auf einem LoggerServer

Durch die Implementierung des LoggerServers wird pro Verbindung ein neuer Thread erstellt. Das Handling der logMessages wird anschliessend an LogHandler übergeben. Somit wird gewährleistet, dass der Server mehrere Verbindungen gleichzeitig Handeln kann und immer ansprechbar ist.

Bei mehreren offenen Verbindungen muss beim Persistieren von LogMessages der Absender auch abgespeichert werden. In unserem Fall lösen wir dies, indem wir pro Client ein spezifisches File erstellen und darin die LogMessages schreiben.

Getestet wird die gesamte Funktionalität manuell. Auf einen automatisierten Testfall wurde hier verzichtet, da es sich bereits um einen kompletten Systemtest handelt. Die einzelnen Teile werden jedoch mittels Unit-Tests getestet.
Die Methode testCreateNewFile() testet, ob ein neues File erstellt werden kann. Mittels testUseExistingFile() wird getestet, ob ein bestehendes File genutzt werden kann.

In einem Extremtest werden 1000 Clients auf denselben Server verbunden. In diesem Fall sollte der Server einen Error zurückliefern, da es ansonsten zu einem Stackoverflow kommen kann.

4.2.7. Dauerhafte Speicherung von Messages

Vom Client erhält der Server ein serialisiertes LogMessage-Objekt. Es wird auf serverseite deserialisiert. Dieses hat die Attribute message, timestamp und logLevel.
Mittels Interface Stringpersistor werden die Daten in ein spezifisches File für den jeweiligen Client geschrieben. Zusätzlich wird noch der Timestamp des Erhalts vom Server hinzugefügt.
Diese Werte werden dann via Outputstream in das File geschrieben.
Getestet wird dies mittels Unit-Tests. Das Erstellen sowie nutzen von bestehenden Files wurde weiter oben bereits beschrieben. Die Methoden testReadFromFile(), testWriteToFile(), testen, ob die Daten

korrekt in das File geschrieben wurden, bzw. auch wieder korrekt ausgelesen werden konnten. Des Weiteren wird mittels testAttributesCorrect() getestet, ob alle Attribute vorhanden sind.

4.2.7.1. Benchmarks

Mittels der Methode testStringPersistorWriteBenchmark() wird getestet, ob der StringPersistor 1000 Nachrichten innerhalb kurzer Zeit zuverlässig in ein File schreiben kann. Dazu wird anschliessend mittels assert die Anzahl Zeilen im File mit 1000 verglichen. testStringPersistorWriteBenchmarkTime() misst danach mit einer Stopwatch die Zeit, welche dafür benötigt wurde. Diese muss kleiner als 1 Sekunden sein. Die Methode testStringPersistorReadBenchmark() liest 500 Zeilen in eine List ein. Die Anzahl dieser List wird anschliessend mittels assert verglichen. testStringPersistorReadBenchmarkTime() misst die Zeit. Auch hier sollte die Zeit wieder kleiner als 1 Sekunde sein.

4.2.8. StringPersistor und StringPersistorFile

Das Interface StringPersistor wird von der API zur Verfügung gestellt und von StringPersistorFile implementiert. Die überschriebenen Methoden werden dabei mit einfachen Unit-Tests getestet. Das anzuwendende Adapter Pattern ist im UML Diagramm klar erkennbar. Der StringPersistor arbeitet Intern mit der Instanzen von PersistedString. Für den Austausch von Messages wurde das Interface *StringPersistorAdapter* erstellt. Der Server implementiert dieses Interface. Die Klasse StringPersistorFile konvertiert die Message in ein für den StringPersistor verständliches Format, gemäss Doku als String.

Dies wird mittels Unit-Tests getestet, indem ein log im jeweiligen File gespeichert wird und dieser mittels assert identisch mit dem abgesendeten ist.

4.2.9. Lokales Zwischenspeichern

Da auf keinen Fall logs verloren gehen dürfen, werden diese auch auf dem LoggerComponent selbst in einer List sowie in einem File gespeichert. Um diese auch über die Laufzeit hinaus zu erhalten, werden diese in ein File geschrieben. Hierzu wird wieder der StringPersistor verwendet. Der Testfall für diesen ist identisch mit obigem.

Bei unserer Implementierung werden alle logs auch lokal gespeichert. Durch ein Flag wird gekennzeichnet, ob eine LogMessage erfolgreich übertragen wurde. Ist dies nicht der Fall, wird sie erneut gesendet.
Dieser Mechanismus wird mittels testResendMessage() getestet. Hierbei wird eine Message aus der List mit dem Flag send:false gekennzeichnet. Sie muss also nocheinmal gesendet werden. testAlreadySendMessage() testet das Gegenteil. Es soll nichts mehr gesendet werden. Mit testSetFlag(), testGetFlag() wird der getter und setter getestet.

4.3. Testcases

Testcase 1 - Server nicht verfügbar. Logger soll lokal loggen können:

Erwartetes Resultat: Falls Server nicht verfügbar ist, soll lokal geloggt werden.

Testcase 2 - Server ist vorübergehend nicht verfügbar. Lokal geloggte Messages sollen nachgesendet werden, sobald Server verfügbar:

Erwartetes Resultat: Falls Server nicht verfügbar ist, wird lokal geloggt. Sobald dieser wieder verfügbar ist, werden Messages nachgesendet.

Testcase 3 - Server verfügbar. Normales Loggen:

Erwartetes Resultat: Normales loggen funktioniert, wenn der Server verfügbar ist.

Testcase 4 - ConfigFile nicht verfügbar auf Game. Es soll ein neues mit Standard erstellt werden.

Erwartetes Resultat: Es wird ein neues ConfigFile erstellt, falls nicht auf Game verfügbar.

Testcase 5 - Applikation soll ohne IDE laufen

Erwartetes Resultat: Applikation lässt sich ohne IDE ausführen.

Testcase 6 - LoggerViewer zeigt Daten bei Erhalt von Logs an

Erwartetes Resultat: LoggerViewer zeigt die Daten bei Erhalt von Logs an

4.4. Testübersicht

4.4.1. Testabdeckung

S	W	Name ↓	Last Version	Git Branches	# Checkstyle	# PMD	Zeilenabdeckung	Letzter Erfolg	Letzter Fehlschlag	Letzte Dauer	Cause
✓	☀	g00-loggerinterface	1.0.0	**	0	0	100.0%	1 Monat 9 Tage - #24	2 Monate 3 Tage - #7	41 Sekunden	
✓	☀	g05-game	1.0.0	**	263	32	0.96%	22 Stunden - #148	1 Monat 8 Tage - #75	1 Minute 8 Sekunden	
✓	☁	g05-logger	2.0.0	**	115	23	50.48%	6 Minuten 19 Sekunden - #181	13 Minuten - #180	1 Minute 37 Sekunden	
✓	☀	g05-stringpersistor	1.4.0	**	114	15	68.83%	8 Tage 8 Stunden - #38	16 Tage - #27	1 Minute 34 Sekunden	

Gemäss Jenkins erreichen wir mit unseren Komponenten folgende Testabdeckungen

Für das Projekt GameOfLife wurde auf die Implementierung von Testfällen verzichtet. Dies ist nicht Gegenstand dieses Projekts.

5. Testdurchführung

5.1. Testcase 1

Eingetreten: Eingetreten ist das erwartete Resultat.

Massnahmen: keine Massnahmen nötig.

5.2. Testcase 2

Eingetreten: Eingetreten ist das erwartete Resultat.

Massnahmen: keine Massnahmen nötig.

5.3. Testcase 3

Eingetreten: Eingetreten ist das erwartete Resultat.

Massnahmen: keine Massnahmen nötig.

5.4. Testcase 4

Eingetreten: Eingetreten ist das erwartete Resultat.

Massnahmen: keine Massnahmen nötig.

5.5. Testcase 5

Eingetreten: Konnte aufgrund von Problemen mit dem HSLU-Nexus-Repository nicht getestet werden

Massnahmen: Wiederholt sobald das Repository wieder funktionierte. Alles wie erwartet.

5.6. Testcase 6

Eingetreten: Eingetreten ist das erwartete Resultat.

Massnahmen: keine Massnahmen nötig.

6. Anhänge

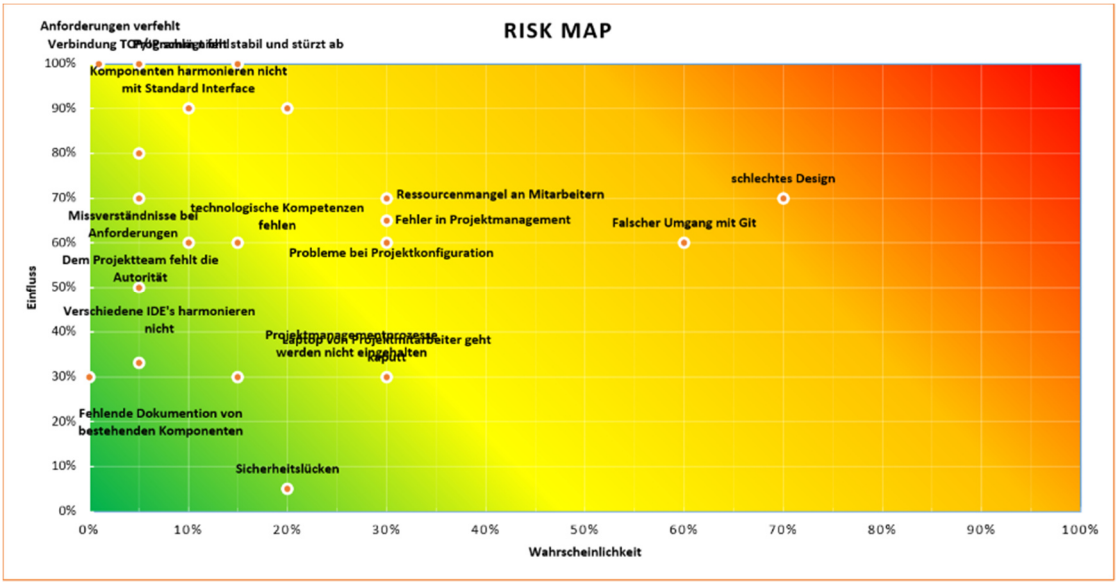
6.1. Projektstrukturplan

Projektstrukturplan

Projektleiter:	Majkel Zivkovski
Datum:	30. Sep. 19

Message Logger			
Vorprojektphase	**Initialisierungsphase**	**Konzeptionsphase**	**Einführungsphase**
Organisation	Interface Komitee	Rollende Detail-Planung	1. Release
Codeanalyse	Rahmenplan	Testplan	2. Release
Dokumentation	ProductBacklog	Entwicklung	
		Testdruchführung	
		Risikomanagement	

6.2. Riskomatrix



6.3. Product Backlog

Product Backlog

Autor:	Frederico Fischer
Stichtag:	3. Okt. 19

Message Logger			
Stories	**Priorität**	**Geschätzer Zeitaufwand***	**Sprint**
Code von Game of Life analysieren	-	2	-
Entscheid für Interface	-	1	-
Interface analysieren	Hoch	1	1
Logger-Komponente als Komponente realisieren	Hoch	3	1
Bei jedem Log-Eintrag hat die aufrufende Applikation einen Message-Level mitzugeben. Dieser Level kann zur Laufzeit geändert werden.	Hoch	1	1
Logger Sever designen und realisieren	Hoch	3	1
TCP/IP Kommunikation zwischen Server und Logger einrichten	Mittel	3	1
Aufzeichnung von Log-Ereignissen garantieren	Mittel	1	2
Die Logger-Komponente ist austauschbar und plattformunabhängig zu realisieren.	Mittel	1	1
paralleles loggen von mehreren Instanzen der Logger-Komponente ermöglichen	Mittel	1	2
Die Komponente StringPersistor designen und realisieren	Tief	1	2
Das Textfile enthält mindestens die Quelle der Logmeldung zwei Zeitstempel	Tief	1	2
Die dauerhafte Speicherung der Messages erfolgt auf dem Server in einem Textfile	Tief	1	2
Adapter (GoF-Pattern) verwenden, um Daten in strukturierter Form in den Payload Parameter der StringPersistor Schnittstelle übergeben zu können.	Tief	1	2

* In Tagen

6.4. Definition of Done

Die Definition of Done (DoD) ist ein gemeinsames Verständnis des Scrum Teams, unter welchen Bedingungen eine Arbeit als fertig bezeichnet werden kann. Sie enthält für gewöhnlich Qualitätskriterien, Einschränkungen und allgemeine nicht-funktionale Anforderungen. Mit zunehmender Erfahrung des Scrum Teams entwickelt sich die Definition of Done weiter. Sie enthält dann strengere Kriterien für höhere Qualität (Wikipedia, 2019). Im Rahmen des VSK-Projektes wurden folgende DoD's formuliert:

- Die Funktionalität wurde vom Product Owner überprüft
- Es wurden Unit Tests durchgeführt und sie sind auf "Grün"
- Build auf Jenkins funktioniert
- Es wurde ein Codereview durchgeführt

- Dokumentation ist für Release 1 auf aktuellem Stand
- Dokumentation ist für Release 2 auf aktuellem Stand

6.5. Sprintplan 1

1	Interface Logger analysieren	Sprint 1
2	Interface LoggerSetup analysieren	Sprint 1
3	Unit Tests Logger erstellen	Sprint 1
4	Unit Tests LoggerSetup	Sprint 1
5	Logger - Methoden implementieren	Sprint 1
6	LoggerSetup Methoden implementieren	Sprint 1
7	Logger testen	Sprint 1
8	LoggerSetup testen	Sprint 1
9	LoggerSetup testen	Sprint 1
10	UnitTests für Logger Komponente mit MessageLevel	Sprint 1
11	MessageLevels mit der Logger Komponente realisieren	Sprint 1
12	MessageLevels mit der Logger Komponente testen	Sprint 1
13	Logger Server designen	Sprint 1
14	UnitTests Logger-Server erstellen	Sprint 1
15	Logger-Server entwickeln	Sprint 1
16	Logger Server testen	Sprint 1
17	Sockets für Logger Server und Logger Komponente desginen	Sprint 1
18	UnitTests für Sockets (Komponente und Server) erstellen	Sprint 1
19	Sockets für Logger Komponente und Server realisieren	Sprint 1
20	Sockets für Logger Komponente und Server testen	Sprint 1
21	DesignPattern überprüfen und testen	Sprint 1
22	Design für StringPersistor mit LoggerServer erstellen	Sprint 1
23	Start der Dokumentation	Sprint 1

6.5.1. Sprintreview 1

Im Nachfolgenden werden zwei Listen aufgeführt, welche den Issues den Status done oder doing zuordnen:

Fertiggestellte Issues:

Issue ID	Title	Milestone	Status
1	Interface Logger analysieren	Sprint 1	done
2	Interface LoggerSetup analysieren	Sprint 1	done
5	Logger - Methoden implementieren	Sprint 1	done
6	LoggerSetup Methoden implementieren	Sprint 1	done
11	MessageLevels mit der Logger Komponente realisieren	Sprint 1	done
13	Logger Server designen	Sprint 1	done
15	Logger-Server entwickeln	Sprint 1	done
17	Sockets für Logger Server und Logger Komponente desginen	Sprint 1	done
19	Sockets für Logger Komponente und Server realisieren	Sprint 1	done
22	Design für StringPersistor mit LoggerServer erstellen	Sprint 1	done
23	Start der Dokumentation	Sprint 1	done

Nicht fertiggestellte Issues:

Issue ID	Title	Milestone	Status
3	Unit Tests Logger erstellen	Sprint 1	doing
4	Unit Tests LoggerSetup	Sprint 1	doing
7	Logger testen	Sprint 1	doing
8	LoggerSetup testen	Sprint 1	doing
9	LoggerSetup testen	Sprint 1	doing
10	UnitTests für Logger Komponente mit MessageLevel	Sprint 1	doing
12	MessageLevels mit der Logger Komponente testen	Sprint 1	doing
14	UnitTests Logger-Server erstellen	Sprint 1	doing
16	Logger Server testen	Sprint 1	doing
18	UnitTests für Sockers (Komponente und Server) erstellen	Sprint 1	doing
20	Sockets für Logger Komponente und Server testen	Sprint 1	doing
21	DesignPattern überprüfen und testen	Sprint 1	doing

Wie aus den beiden Grafiken ersichtlich, konnten währen des ersten Sprints nicht alle Issues fertiggestellt werden. Die nicht fertiggestellten Issues umfassen vollumfänglich den Bereich Testing und müssen während des zweiten Sprints nachgeholt werden. Es gibt diverse Ursachen weshalb nicht alle Issues abgeschlossen wurden. Zum einen hat die Einarbeitung in die Thematik mehr Zeit gekostet als erwartet. Zum anderen benötigte das Aufsetzen der Umgebung und die Einarbeitung in die Tools ebenfalls eine gewisse Zeit.

6.6. Sprintplan 2

24	StringPersistor Interface-Methoden implementieren	Sprint 2
25	StringPersistorFile designen	Sprint 2
26	StringPersistorFile implementieren	Sprint 2
27	Unit Tests StringPersistor erstellen	Sprint 2
28	Unit Tests StringPersistorFile erstellen	Sprint 2
29	StringPersistor testen	Sprint 2
30	StringPersistorFile testen	Sprint 2
31	Problems with createLoggerSetup from factory	Sprint 2
32	Implementierung Adapter Pattern	Sprint 2
33	StringPersistorAdapter designen, realisieren und testen	Sprint 2
34	Dokumentation nachführen	Sprint 2
35	Konfigurationsfile sollte eingelesen werden können	Sprint 2
36	Log-Einträge im GoL	Sprint 2
37	Exceptionhandling	Sprint 2
38	Tests gemäss Testplan	Sprint 2
39	Server soll einen Messageerhalt bestätigen	Sprint 2
40	Message erneut senden, falls senden fehlschlägt	Sprint 2

6.6.1. Sprintreview 2

Im Nachfolgenden wird eine Liste aufgeführt, welche den Issues den Status done zuordnet:

Issue ID	Title	Milestone	Status
24	StringPersistor Interface-Methoden implementieren	Sprint 2	done
25	StringPersistorFile designen	Sprint 2	done
26	StringPersistorFile implementieren	Sprint 2	done
27	Unit Tests StringPersistor erstellen	Sprint 2	done
28	Unit Tests StringPersistorFile erstellen	Sprint 2	done
29	StringPersistor testen	Sprint 2	done
30	StringPersistorFile testen	Sprint 2	done
31	Problems with createLoggerSetup from factory	Sprint 2	done
32	Implementierung Adapter Pattern	Sprint 2	done
33	StringPersistorAdapter designen, realisieren und testen	Sprint 2	done
34	Dokumentation nachführen	Sprint 2	done
35	Konfigurationsfile sollte eingelesen werden können	Sprint 2	done
36	Log-Einträge im GoL	Sprint 2	done
37	Exceptionhandling	Sprint 2	done
38	Tests gemäss Testplan	Sprint 2	done
39	Server soll einen Messageerhalt bestätigen	Sprint 2	done
40	Message erneut senden, falls senden fehlschlägt	Sprint 2	done

Der Sprint 2 verlief erfolgreicher als der Sprint 1, denn es konnten alle geplanten Issues erfolgreich beendet werden. Zudem war es möglich den Rückstand im Testing Bereich aus dem ersten Sprint aufzuholen. Somit konnten alle Anforderungen für den ersten Release erfolgreich abgeschlossen werden.

6.7. Sprintplan 3

41	Test Multiuser	Sprint 3
42	String Persitor Client	Sprint 3
43	Quality of String Persistor	Sprint 3
45	Configuration File	Sprint 3
46	GUI	Sprint 3
49	More Logs Game of Life	Sprint 3
52	Refactor tests to map the presentation from friday	Sprint 3
54	StrategySaveSerializedObject	Sprint 3
55	Logger Testing on Game	Sprint 3
57	Factory	Sprint 3
67	improve riskmanagement	Sprint 3

6.7.1. Sprintreview 3

Im Nachfolgenden wird eine Liste aufgeführt, welche den Issues den Status done zuordnet:

Issue ID	Title	Milestone	Status
41	Test Multiuser	Sprint 3	done
42	String Persitor Client	Sprint 3	done
43	Quality of String Persistor	Sprint 3	done
45	Configuration File	Sprint 3	done
46	GUI	Sprint 3	done
49	More Logs Game of Life	Sprint 3	done
52	Refactor tests to map the presentation from friday	Sprint 3	done
54	StrategySaveSerializedObject	Sprint 3	done
55	Logger Testing on Game	Sprint 3	done
57	Factory	Sprint 3	done
67	improve riskmanagement	Sprint 3	done

Der dritte Sprint verlief ebenso erfolgreich wie der zweite Sprint und es konnten alle Issues erfolgreich abgeschlossen werden.

6.8. Sprintplan 4

44	Strategy pattern	Sprint 4
47	RMI	Sprint 4
48	Improve Documentation	Sprint 4
51	CodeCoverage of the project StringPersistor	Sprint 4
53	Refactore Strategy to store date different formats	Sprint 4
56	running outside IDE	Sprint 4
58	Review	Sprint 4
59	Write and test some testcases	Sprint 4
60	Documentation Teststrategy	Sprint 4
61	Documentation Adapter Pattern	Sprint 4
62	Documentation Strategy Pattern	Sprint 4
63	Documentation LogFile	Sprint 4
64	Documentation ConfigFile	Sprint 4
65	Documentation last discussion	Sprint 4
66	grafical "Rahmenplan"	Sprint 4
68	add acceptance criteria	Sprint 4
69	reference to Interface specification	Sprint 4
70	improve documentation TCP/IP	Sprint 4
71	Add configFile to LoggerServer	Sprint 4

6.8.1. Sprintreview 4

Im Nachfolgenden wird eine Liste aufgeführt, welche den Issues den Status done zuordnet:

Issue ID	Title	Milestone	Status
44	Strategy pattern	Sprint 4	done
47	RMI	Sprint 4	done
48	Improve Documentation	Sprint 4	done
51	CodeCoverage of the project StringPersistor	Sprint 4	done
53	Refactore Strategy to store date different formats	Sprint 4	done
56	running outside IDE	Sprint 4	done
58	Review	Sprint 4	done
59	Write and test some testcases	Sprint 4	done
60	Documentation Teststrategy	Sprint 4	done
61	Documentation Adapter Pattern	Sprint 4	done
62	Documentation Strategy Pattern	Sprint 4	done
63	Documentation LogFile	Sprint 4	done
64	Documentation ConfigFile	Sprint 4	done
65	Documentation last discussion	Sprint 4	done
66	grafical "Rahmenplan"	Sprint 4	done
68	add acceptance criteria	Sprint 4	done
69	reference to Interface specification	Sprint 4	done
70	improve documentation TCP/IP	Sprint 4	done
71	Add configFile to LoggerServer	Sprint 4	done

Der letzte Sprint verlief ebenso erfolgreich wie die letzten beiden, denn es konnten alle geplanten Issues erfolgreich beendet werden. Dadurch konnten alle Anforderungen für den zweiten Release erfolgreich abgeschlossen werden.

6.9. Meilensteinberichte

6.9.1. Meilensteinbericht 1

6.9.1.1. Termin Meilenstein 1

Der erste Meilenstein findet zu Beginn der dritten Semesterwoche statt.

6.9.1.2. 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 Frederico Fischer sein. Neben der Präsentation soll die Organisation im Team koordiniert werden.

6.9.1.3. Meilensteinziele/-vorgaben

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

6.9.1.4. Meilensteinzielerreichung

1. Der Delegierte Frederico Fischer hat eine Variante eines Interface-Sets präsentiert
2. Nach den drei Präsentation wurde eine erfolgreiche Diskussion geführt
3. Das Plenum hat eine Entscheidung für ein Interface-Set gefällt.
4. PMP Dokumentation gestartet
    a. SoDa-Rollen sind definiert
    b. Risikoliste ist erstellt
    c. Product Backlog ist definiert
    d. Die erste Sprintplanung steht

6.9.2. Meilensteinbericht 2

6.9.2.1. Termin Meilenstein 2

Der zweite Meilenstein findet zu Beginn der achten Semesterwoche statt.

6.9.2.2. 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.

6.9.2.3. Anforderungen an das Message Logging Systems

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.

6.9.2.4. Meilensteinziele

1. Ein Logger-Interface zur Verfügung stellen
2. Logger-Komponente implementieren

3. Logger-Server implementieren
4. StringPersistor-Komponente implementieren
5. Events innerhalb von GoL über das Logger-Interface mit der Logger-Komponente auf dem Logger-Server mittels StringPersistor loggen
6. Vollständige Dokumentation

6.9.2.5. Meilensteinzielerreichung

1. 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 GoL werden auf dem Logger-Server geloggt
6. Die Dokumentation ist vollständig

6.9.3. Meilensteinbericht 3

6.9.3.1. Termin Meilenstein 3

Der dritte Meilenstein findet in der 13. Semesterwoche statt.

6.9.3.2. Beschreibung Meilenstein 3

Bei dem 3. Meilenstein geht es um die endgültige Abgabe des Message Logging Systems. Zudem muss die Lösung im Plenum präsentiert werden.

6.9.3.3. Anforderungen an das Message Logging Systems

2.4. Muss-Features bei Schlussabgabe

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).

6.9.3.4. Meilensteinziele

1. Präsentation der eigenen Lösung
2. Demo der Software mit eigener Logger-Komponente
3. Demo der Integration einer fremden Logger-Komponente
4. Review durchgeführt und protokolliert
5. Vollständige Dokumentation, Release 1 und 2
6. Ausserhalb der IDE lauffähige Programme

6.9.3.5. Meilensteinzielerreichung

1. Lösung kann präsentiert werden
2. Demo der Software mit eigener Logger-Komponente ist möglich
3. Demo der Integration einer fremden Logger-Komponente ist möglich
4. Review wurde durchgeführt und protokolliert
5. Vollständige Dokumentation, Release 1 und 2 ist abgeschlossen
6. Programme können ausserhalb der IDE ausgeführt werden