Gruppe 6

Modul VSK 1801

Message-Logger

Projektmanagement-Plan

Änderungsprotokoll

Rev.	Datum	Autor	Bemerkungen	Status
0.0.1	12.03.2018	Projektleiter	1. Entwurf	Fertig
0.2.0	17.03.2018	Projektleiter	2. Entwurf	Fertig
0.3.0	20.03.2018	Projektleiter	3. Entwurf	Fertig
0.3.1	23.03.2018	Projektmitarbeiter	Review	Fertig
1.0.0	25.03.2018	Projektleiter	Release 1.0.0	Fertig
1.0.1	27.03.2018	Projektleiter	Add Rahmenplan, Strukturplan	Fertig
1.0.2	27.03.2018	Product Owner	Abnahme Sprint 1	Fertig
1.0.3	09.04.2018	Product Owner	Abnahme Sprint 2	Fertig

Inhaltsverzeichnis

1. Projektorganisation

1.1. Organisationsplan, Rollen & Zuständigkeiten

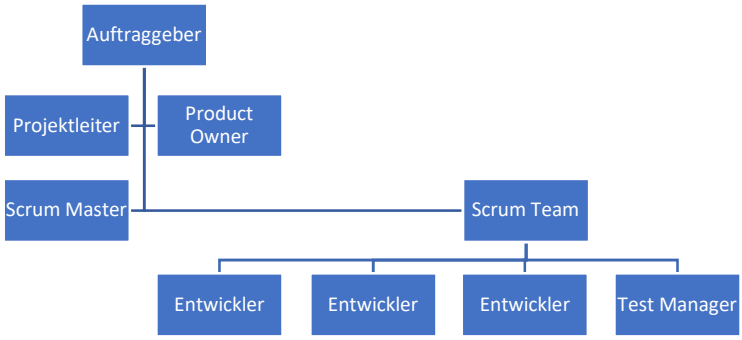


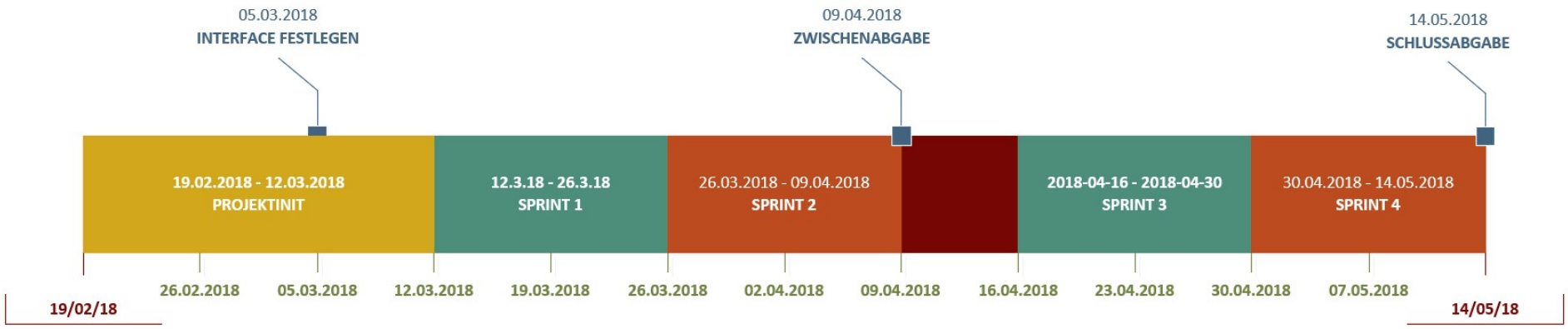
Abbildung 1: Organisation

1.1.1. Mitglieder

Mitglied	**Rollen**
Tobias Jaeggi	Scrum Master, Abgeordneter Interface
Tim Bolzern	Projektleiter, Entwickler
Pascal Keusch	Entwickler, Test Manager
Roman Schraner	Product Owner, Entwickler

Tabelle 1: Mitglieder

1.2. Projektstrukturplan



2. Projektführung

2.1. Rahmenplan

Der Rahmenplan basiert auf den erfassten Stories und der Planung dieser in ScrumDo.
Dazu folgend die Übersicht über alle bereits vordefinierten Stories und deren Aufwände (Points).

Iteration	Card ID	Summary	Detail	Points
Backlog	V2-11	Robuste Implementation der Socketverbindung	• Als Benutzer mochte ich, dass die Kommunikation zwischen Logger und Log-Server auch Netzwerkunterbruche ohne Verluste uberstehen kann, damit die Logs nicht verfalscht werden.	?
Backlog	V2-4	Konfigurationsanpassungen	• Als Benutzer mochte ich statische Konfigurationsdaten der Komponenten ohne Programmierung anpassen konnen, um so eine leichtere Bedienung zu ermoglichen.	?
Backlog	V2-12	Log Viewer	• Als Benutzer mochte ich, dass ein Log-Viewer vorhanden ist, mit dem die Logs dargestellt werden konnen, damit eine einfache Analyse der Logs moglich ist.	?
Sprint-1	V2-1	Log Events in Levels unterteilen	• Als Benutzer mochte ich, dass einem Log Eintrag ein Message Level (Log Level) mitgegeben werden kann, damit ich die Log Eintrage einschranken kann. Abnahmekriterien: • Es existieren definierte MessageLevels, welche alle vom Game aus erreicht werden konnen.	3
Sprint-1	V2-14	Einheitliches LogEvent	• Als Entwickler mochte ich, dass ein einheitliches Log-Event auf der Client-	8

			Logger-Komponente und dem Logger-Server vorhanden ist, um die TCP Kommunikation einfach zu halten (serialisieren, deserialisieren). Abnahmekriterien: • Es besteht eine LogEvent Klasse im loggercommon Modul, welches vom Client und vom Server fur die Kommunikation verwendet wird.	
Sprint-1	V2-7	Message-Level wahrend Laufzeit andern	• Als Benutzer mochte ich, dass ich wahrend der Laufzeit (des Loggers) das Message-Level andern kann, sodass bei Bedarf mehr oder weniger geloggt wird. * Als Benutzer mochte ich, dass das Message-Level im Game definiert werden kann, damit ich dynamisch das Level setzen kann Abnahmekriterien: • Im GameOfLife besteht die Moglichkeit das MessageLevel uber das GUI zu setzen.	5
Sprint-1	V2-6	Logger Komponente als Interface	• Als Benutzer mochte ich, dass die Logger Komponente als Interface implementiert ist, sodass diese ohne Code Anpassungen austauschbar ist. Abnahmekriterien: • Es besteht ein Interface fur die Logger Komponente. • Das Interface wird von der Logger Komponente korrekt implementiert.	8

Sprint-1	V2-5	Implementierung Log-Persistor	• Als Logger-Server mochte ich das Log-Persistor Interface implementieren, um die Log Messages persistent ablegen zu konnen. Abnahmekriterium: • Eine Log Message kann vom Logger Server in ein File persistiert werden.	8
Sprint-2	V2-8	Unterteilung Logger und LoggerSetup	• Als Benutzer mochte ich, dass die Logger-Komponente mindestens in ein Logger (Message erzeugen und eintragen) und LoggerSetup (Konfiguration des Loggers) unterteil ist, sodass ich dynamisch einen Logger generieren und verwalten kann. Abhnamekriterien: • Die Logger-Komponente kann ausgetauscht werden, ohne das Code Anpassungen im GameOfLife durchgefuhrt werden mussen.	3
Sprint-2	V2-16	Game Logs definieren	• Als Benutzer mochte ich, dass im Game of Life sinnvolle Log-Eintrage definiert sind, sodass im Log sinnvolle Eintrage erfasst sind. Abnahmekriterien: • Im Game sind Log Eintrage definiert.	2
Sprint-2	V2-2	Paralleles Logging	• Als Benutzer mochte ich auf dem zentralen Server paralleles Logging erlauben, damit mehrere Instanzen gleichzeitig loggen konnen.	8

Sprint-2	V2-15	Server-Komponente Multi-Threaded	• Als Entwickler mochte ich, dass der TCP Listener standig neue Clients akzeptieren kann (Thread erzeugen, wenn eine Anfrage kommt), sodass simultanes Logging möglich ist. Abnahmekriterien: • Es konnen mehrere Instanzen des GameOfLife ausgefuhrt werden und Loggen.	3
Sprint-2	V2-9	Logger arbeitet zuverlassig	• Als Benutzer mochte ich, dass alle Logs in der korrekten Reihenfolge (Zeitstempel, logische Abfolge) persistiert werden, sodass ich die Logs unverfalscht anschauen kann.	5
Sprint-2	V2-3	Dauerhafte Speicherung (mit Adapter)	• Als Benutzer mochte ich, dass die Log Messages dauerhaft in einem einfachen lesbaren Textfile mit Quelle, zwei Zeitstempel, Message Level und Message-Text abgespeichert werden, damit Messages auch zu einem spateren Zeitpunkt noch eingesehen werden konnen. Abnahmekriterium: • Die Log-Eintrage sind korrekt (korrekte Reihenfolge, Zeitstempel, Message, Level und Exception).	8

2.2. Projektkontrolle

Für das Projektcontrolling kommen folgende Werkzeuge/Instrumente zum Einsatz:

- Daily Meeting
- Burndown Chart
- Sprint Review und Retrospektive

Die detaillierten Controlling-Elemente der jeweiligen Sprints ist im Anhang ersichtlich.

2.2.1. Definition of Done

Ein Sprint gilt als abgeschlossen, wenn folgende Vorgaben erfüllt sind:

- Alle vorgesehenen User Stories eines Sprints sind abgeschlossen.
- Die Programmierten Komponenten wurden anhand von Unit-Tests getestet.
- Die Integrationstests verliefen erfolgreich.
- Auf dem Build-Server können alle Builds erfolgreich (ohne Fehler) generiert werden.
- Der Product Owner hat den Abnahmetest der «potentially shippable» Komponente durchgeführt und abgenommen.
- Die Komponenten sind Dokumentiert.

2.2.2. Aufwandschätzung

Die Aufwandschätzung der User Stories wird jeweils vor dem Sprint durch das Scrum Team durchgeführt. Zuerst werden die nötigen Tasks für die entsprechende User Story definiert und anschliessend gibt jedes Team-Mitglied eine Aufwandschätzung ab.
Die Aufwandschätzung enthält jeweils den Aufwand für Design, Implementierung und Testing. Der Durchschnitt wird als SOLL Aufwand in der Planung verbucht.
Die Aufwände werden in Points definiert. Jedes Team-Mitglied hat pro Sprint ca. 8 Points Arbeitszeit zur Verfügung.

2.2.3. Entwicklungsrichtlinien

Entwickler sollte folgende Richtlinien beachten:

- für jede User Story ein Branch erstellen (Name gemäss Story ID in ScrumDo)
- die aufgewendete Zeit pro User Story zu vermerken
- Unittests implementieren

2.3. Risikomanagement

2.3.1. Risikoeinteilung

2.3.1.1. Eintrittswahrscheinlichkeit

Stufe	Definition
5	Alles deutet darauf hin
4	Grosse Wahrscheinlichkeit
3	Gleichverteilte Chance für den Eintritt
2	Manchmal tritt das Problem ein
1	Sehr unwahrscheinlich

Tabelle 2: Eintrittswahrscheinlichkeit

2.3.1.2. Schadensausmass

Stufe	Definition
5	Katastrophal
4	Kritisch
3	Mittelmässig
2	Gering
1	vernachlässigbar

Tabelle 3: Schadensausmass

2.3.2. Risiken

EW= Eintrittswahrscheinlichkeit
SA = Schadensausmass

ID	Risiko	Auswirkungen
1	Ausfall eines Teammitglieds	• Verzögerungen in der Realisierung des Projekts
2	Git-Server Absturz	• Verlust des aktuellsten Entwicklungsstandes • Projektverzögerungen
3	Fehler des Logger-Interface	• Mehraufwand an anderen Komponenten
4	Fehlerhafter Zeitplan	• Projektverzögerungen
5	Projekt-Changes (neue Anforderungen)	• Mögliche Projektverzögerungen
6	Fehlendes Know-How	• Zusätzlicher Arbeitsaufwand • Mögliche Projektverzögerungen
7	Internet Absturz	• Erschwerte Arbeit
8	Fehlinterpretationen der Anforderungen	• Unvollständiger Projektumfang • Ggf. Projektverzögerungen zur Realisierung fehlen
9	Gesetzesänderungen	• Zusatzaufwände für deren Einhaltung • ggf. Projektverzögerungen
10	Mangelnde Arbeitsleistung	• ggf. Projektverzögerungen • Mehrbelastung für restliche Projektteilnehmer

Tabelle 4: Risiken

2.3.3. Anhä

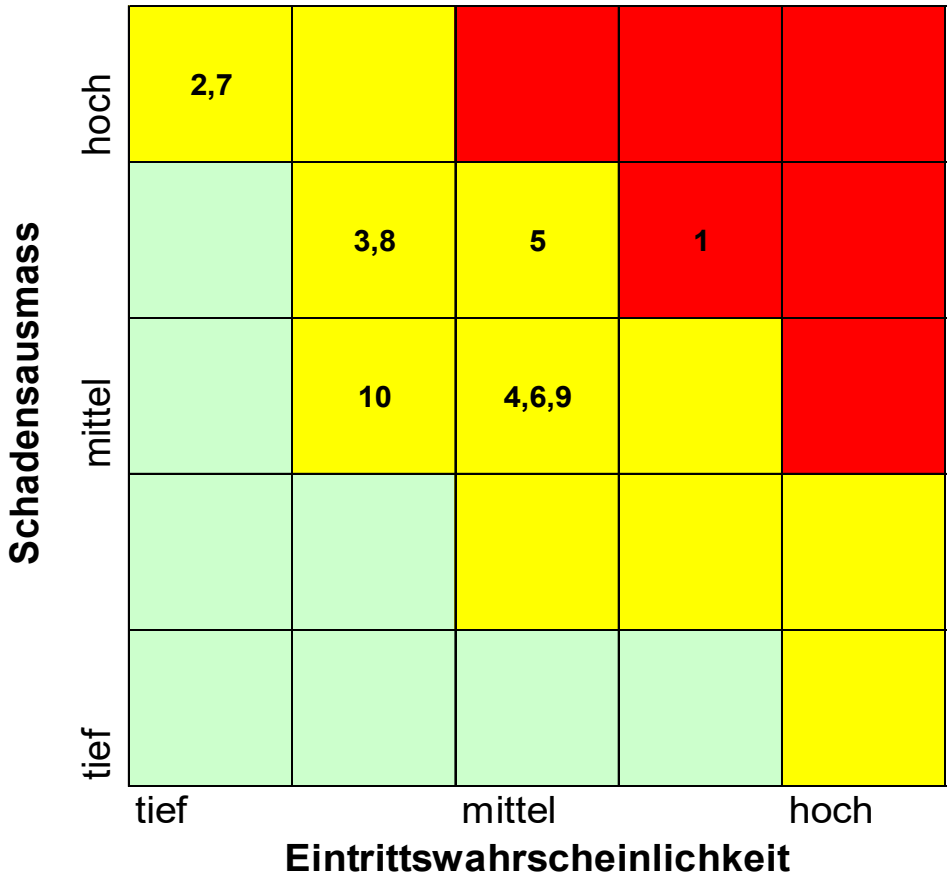


Abbildung 2: Risiko-Matrix

2.3.4. Massnahmenplan

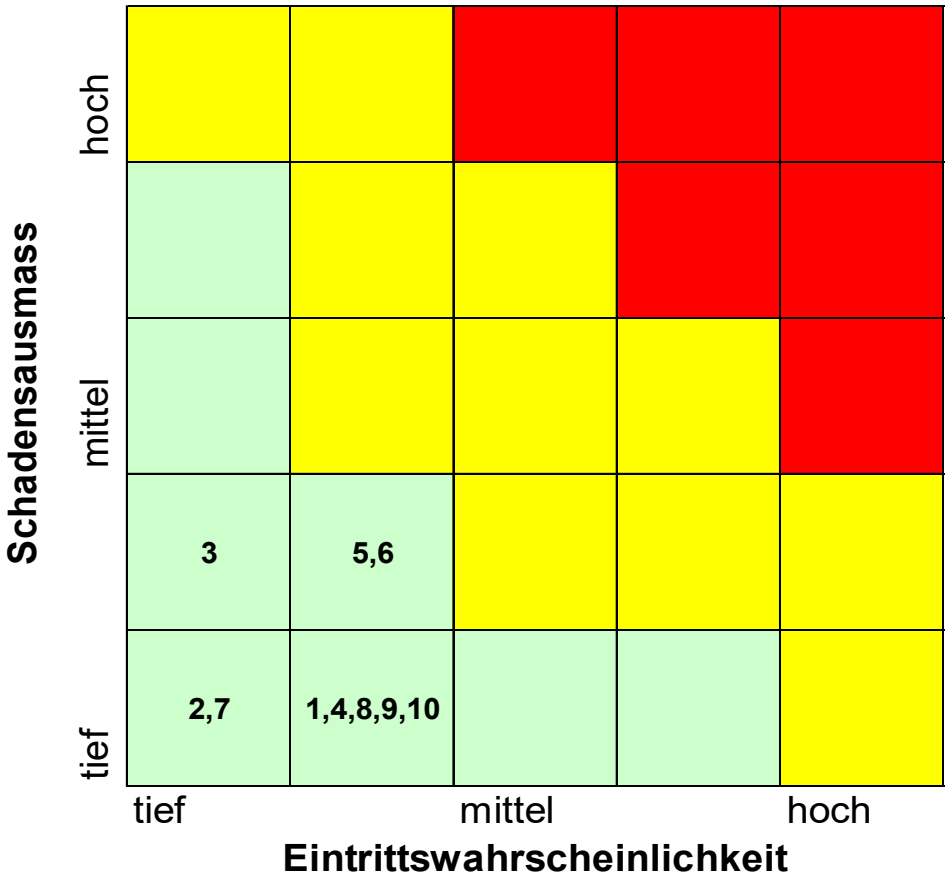
Es werden jeweils Massnahmen definiert um die Eintrittswahrscheinlichkeit und das Schadensausmass zu vermindern.

ID	Massnahme	Betroffenes Risiko	EW neu	SA neu
1	Wöchentlicher Abgleich der Arbeit aller Teammitglieder.	1	4	1
2	Während des Projekts keine Gesundheitshindernden Aktivitäten ausführen.	1	2	4
3	Am Ende jedes Sprints wird ein Backup des ganzen Git-Repository auf dem lokalen Rechner des Scrum Masters gemacht.	2	1	1
4	Planung des Interfaces durch alle Teams, mit Rücksprache aller Mitglieder.	3	1	4

5	Zeit für allfällige Änderungen am Interface einplanen.	3	2	2
6	Design der Komponenten im Team absprechen.	4	2	3
7	Genügend Reserven bei Fehlplanung einplanen.	4	3	1
8	Anforderungen zu Beginn des Projekts analysieren und allfällige Fragen mit dem Auftraggeber klären.	5	2	4
9	In Planung einberechnen, dass nur ca. 80% der User-Stories bekannt sind.	5	3	2
10	Alle Inputs im Unterricht besuchen und hinterfragen.	6	2	3
11	Regelmässiger Austausch der Teammitglieder mit Code-Review um Know-How auszutauschen.	6	3	2
12	Internetzugriff über 2. WLAN-Netz organisieren.	7	1	5
13	Regelmässige Datensicherung auf lokalen Speicher-medien.	7	1	1
14	Anforderungen im Team durchlesen und besprechen, daraus das Design definieren.	8	1	4
15	Durch Reserven im Zeitplan mögliche Änderungen durchführen	8	2	2
16	Abstimmen gehen	9	1	3
17	Regelmässige Absprache mit Politikern	9	1	2
18	Regelmässiger Austausch unter den Projektmitglie-dern (Daily Meeting)	10	1	3
19	Detaillierte Dokumentation der technischen Umset-zung	10	2	2

Tabelle 5: Massnahmen

2.3.5. Risiko-Matrix nach Massnahmen



2.4. Projektabschluss

Nach Projektabschluss sind die Anforderungen wie folgt umgesetzt:

ID	Anforderung	Status
1	Die Logger-Komponente muss als Komponente (mit Interfaces) realisiert werden.	OK
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.	OK
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.	OK
4	Die Log-Ereignisse werden durch Logger-Komponente und den Logger-Server kausal und verlässlich aufgezeichnet.	OK
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.	OK
6	Es muss möglich sein, dass mehrere Instanzen der Logger-Komponente parallel auf den zentralen Logger-Server loggen.	OK
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.	OK
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.	OK

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 Unittests. Die vorgegebene Schnittstelle String-Persistor muss eingehalten werden.	OK
10	Die Qualitätsmerkmale der vorgegebenen Schnittstelle StringPersistor werden erreicht.	OK
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 Unittests.	
12	Statische Konfigurationsdaten der Komponenten (z.B.: Informationen bezüglich Erreichbarkeit des Servers) sind zu definieren und müssen ohne Programmierung anpassbar sein.	
13	Die Socketverbindung zwischen Komponente und Server muss so robust implementiert werden, dass diese mit Netz-werkunterbrüchen umgehen kann.	

3. Projektunterstützung

3.1. Tools für Entwicklung, Test & Abnahme

Tool	Beschreibung
Gitlab	Gitlab wird als CVS eingesetzt
Jenkins	Jenkins dient zur kontinuierlichen Software-Integration von Softwarekomponenten
ScrumDo	ScrumDo unterstützt in der Umsetzung agiler Softwareprojekte
IntelliJ	IntelliJ wird als Entwicklungsumgebung eingesetzt
Slack	Im Team wird Slack als Kommunikationsinstrument verwendet

Tabelle 6: Werkzeuge Entwicklung, Test & Abnahme

3.2. Konfigurationsmanagement

3.2.1. Configuration Items

Beschreibung aller Configuration Items, welche für die Konfiguration, Benutzung und Wartung nötig sind.
<TODO: Versionen eintragen>
<TODO: Tagging der Releases auf Gitlab> → für Release 1 erledigt

CI	R1, Zwischenabgabe	R2, Schlussabgabe
PMP	??????	
Systemspezifikation	1.0.0	
Logger Interface	??????	
Logger Component	1.1.0	
Logger Server	1.1.0	
Logger Common	1.1.0	
Logger Viewer	- (nicht vorhanden)	
StringPersistor	1.1.0	
GameOfLife	1.1.0	
Entwicklungsumgebung		
Maven	3.0.5	3.0.5
IntelliJ IDEA	2018.1	2018.1
Jenkins	2.114	2.114
GitLab	10.5.7	10.5.7

3.2.2. Versionierung

Bei der Versionierung wird gemäss https://semver.org/ vorgegangen, dies gilt für Programmcode und dazugehörige Dokumente.

3.2.3. Dokumentationsplan

Dokument	Beschreibung	Entstehung	Autor
Projektmanagement Plan	• Organisation • Projektführung • Testplan	SW 02 – SW 13	Projektleiter
Zeitplan	• zeitliche Abfolge von Aktivitäten im Rahmen des Projekts	SW 02 – SW 13	Projektleiter
Testplan	• Zeitliche Abfolge der Testfälle im Rahmen des Projekts	SW 02 – SW 06	Test Manager
Test-Protokoll	• Testresultate inkl. Auswertung • Defect-Beschreibung	SW 02 – SW 13	Test Manager
Installationshandbuch	• Dokumentation der Installation des Loggers	SW 02 – SW 13	Entwickler
Benutzerhandbuch	• Dokumentation zur Verwendung des Loggers	SW 02 – SW 13	Entwickler

Tabelle 7: Dokumentationsplan

4. Testmanagement

4.1. Testrollen

Rolle	Beschreibung
Testmanager	Der Testmanager ist für die Festlegung des Testprozesses und die Erstellung des Testkonzepts zuständig.
Testentwickler	Der Testentwickler ist für die Umsetzung der Komponententests verantwortlich.
Product Owner	Der Product Owner ist für die Validierung der Potentially Shippable Software nach jedem Sprint verantwortlich.

Tabelle 8: Testrollen

4.2. Testdesign & Abläufe

Die Aufwände für Erstellung von automatisierten Tests ist als Bestandteil der User Stories vorgesehen.

4.3. Testobjekte

Das Projekt wird in folgende Testobjekte gegliedert

- Game
- Logger-Interface
- Logger-Komponente
- Logger-Server
- Log Persistor

Besonderes Augenmerk wird auf die Zuverlässigkeit der Verbindung zwischen Logger Komponente und Logger-Server gelegt.

4.4. Testmethoden

Teststufe	Testziele	Testart	Metriken	Verantwortung
Code Qualität	Fehler in der Implementierung und Verbesserungen erkennen. Lerneffekt für Entwickler.	Code Review	Keine kritischen Fehler	Projekt (Entwickler)
Komponententest	Test der Funktionalität innerhalb der Komponente	Unit-Test	Keine kritischen Fehler	Projekt (Entwickler)

Systemtest	Workflow der Log-Verarbeitung funktioniert komponentenübegreifend	Integrationstest Schnittstellentests	Keine kritischen Fehler	Projekt (Entwickler)
Abnahmetest / Validierung	Der Kunde prüft, ob seine Anforderungen erfüllt worden sind. Z.B.: Austauschbarkeit des Interfaces ist gewährleistet	Abnahmetest	Keine kritischen Fehler	Product Owner

Tabelle 9: Testmethoden

4.5. Testplan

Nr.	Vorgehen	Erwartetes Ergebnis
1	1. Vom Logger Server ein LogEvent mittels StringPersistorFile persistieren.	LogEvent wird in Log-File abgespeichert.
2	1. Im Game das zu loggende Level einstellen (WARNING). 2. Log Events für jedes Log Level definieren.	Alle LogEvents, welche geloggt werden sollen, werden geloggt (WARNING, INFO)
3	1. Logger Server starten. 2. Game starten.	Game wird gestartet und ein LogEvent wird persistiert.
4	1. Logger Server starten. 2. Game 1 starten. 3. Game 2 starten.	Games werden gestartet und pro Game werden die LogEvents persistiert.
5	1. Logger-Komponenten austauschen. 2. Logger Server starten. 3. Game starten.	Game wird gestartet und ein LogEvent wird persistiert.
6	1. Server starten 2. Viewer starten 3. Game starten	Alle Log-Messages werden im Viewer korrekt dargestellt
7	1. Server starten 2. Game starten 3. Internetverbindung trennen 4. Game stoppen 5. Game starten 6. Internetverbindung wiederherstellen	Der Logeintrag zum Game starten/stoppen wird nachgereicht vom Client an den Server.
8	1. Server starten 2. Ersten Viewer starten	Beide Viewer zeigen Log Messages an

	3. Zweiten Viewer starten	

4.6. Klassifizierung

Nachfolgende Klassifizierung dient, auftretende Fehler einzuordnen.

4.6.1. Fehlerschwere

Severity	**Beschreibung**
Kritisch	Verhindert die Ausführung des Programms.
Schwer	Eine Ausführung des Programmes ist nur mit Einschränkung möglich.
Mittel	Das Programm lässt sich mit Workarounds nutzen.
Kosmetisch	System lässt sich ohne Fehlerwirkungen vollständig nutzen. Beispiele kosmetischer Fehler sind z.B. Rechtschreibefehler.

Tabelle 10: Fehler Fehlerschwere

4.6.2. Dringlichkeit

Dringlichkeit	**Beschreibung**
Sehr dringend (Major)	Behebung der Fehlerwirkung muss schnellstmöglich erfolgen.
Normal (Normal)	Der Fehler muss innert nützlicher Frist behoben werden.
Nicht zeitkritisch (Minor)	Der Fehler kann ohne Zeitdruck gefixt werden.

Tabelle 11: Dringlichkeit Fehler

5. Anhänge