HSLU Hochschule Luzern

Testplan – Logger Implementation

VSK-F20 / Gruppe 4

Abgabetermin 19.04.2020

Projektteam:

Adrian Willi adrian.willi@stud.hslu.ch
Julien Grüter julien.grueter@stud.hslu.ch
Maurizio Hostettler maurizio.hostettler@stud.hslu.ch
Stephan Stofer stephan.stofer@hslu.ch

Inhaltsverzeichnis

1 Einführung

Der vorliegende Testplan beschreibt die Qualitätssicherung im VSK Logger Projekt, entwickelt von der Gruppe 4 im Frühlingssemester 2020 an der Hochschule Luzern.

1.1 Zweck

Der Testplan dient dazu die realisierte Software und deren Komponenten optimal organisiert und systematisch zu testen. Dazu werden nebst den Vorgaben zum Testvorgehen konkrete Testfälle, Szenarien und Abnahmekriterien definiert.

1.2 Umfang

1.2.1 Automatisierte Tests

Unit Tests
Jeder Entwickler stellt sicher, dass die von ihm geschriebene Software auf Stufe Klasse via Unittests geprüft wird. Dabei wird versucht eine Codeabdeckung von 80% zu erreichen.

Integrationstests
Um das Zusammenspiel der verschiedenen Komponenten zu prüfen, werden analog der Unittests auch Integrationstests in JUnit geschrieben, welche automatisch beim Build geprüft werden. Diese umfassen mehrere Klassen und prüfen deren Vernetzte Funktion.

1.2.2 Manuelle Tests

Integrationstest
Erweiterte Integrationstests werden auf manueller Basis durchgeführt. Hierbei liegt der Fokus mehr auf der Interaktion zwischen kompilierten ausführbaren Softwareteilen.

Systemtestes
Diese überprüfen die Funktionen der Software aus der Sicht des Kunden und stellen auf diese Weise sicher, dass die Software den definierten Anforderungen entspricht.

NFA-Tests
Es wird sichergestellt, dass auch die nicht funktionalen Anforderungen korrekt erreicht werden. Dazu gehört die Geschwindigkeit des gebauten Systems und das Verhalten unter hoher Last.

2 Systemübersicht

Für das Spiel "Game of Life" wird eine Loggersoftware entwickelt, welche entsprechende Meldungen des Spiels über das Netzwerk an einen Logger Server zum persistieren schickt. Der Logger besteht entsprechend aus einem Serverteil und einem Client Teil.

Um den Logger für das System austauschbar zu machen, wird dieser gegen ein allgemein gültiges Interface entwickelt. Durch das dynamische Laden der Logger- Implementation, kann die entwickelte Software unter den verschiedenen Gruppen ausgetauscht werden.

2.1 Module

Der Logger besteht aus den Modulen drei Modulen, welche sich den Interfaces StringPersistor und loggerInterface bedienen.

- Logger.common
  An dieser Stelle wird hauptsächlich das Datenformat der Logmessages festgelegt.
- Logger.component
  Stellt den Clientteil dar welcher im Game integriert wird
- Logger.server
  Nimmt die vom Client erstellten Nachrichten entgegen und persistiert diese auf der Harddisk.

3 Abnahme und Testendkriterien

Damit die Software abgenommen werden kann, müssen alle folgend beschrieben Tests und Aspekte entsprechend deren Details erfüllt werden

- CI/CD Pipeline
  Erstes Abnahmekriterium ist der saubere Build über die CI/CD Pipeline. Dies beinhaltet alle für den Build notwendigen Punkte, unter anderem sauber kommentierten Code und ein fehlerfreies Durchlaufen aller im Projekt vorhandenen Tests.
  Auf diesen Teil der Prüfung (Rules auf Jenkins) hat das Entwicklerteam keinen Einfluss

- Manuelle Tests
  Für alle manuell durchgeführten Tests wird ein Testprotokoll erstellt in dem die gefundenen Fehler dokumentiert und kategorisiert werden. Die Software kann nicht abgenommen werden, solange noch Fehler der Klassen 2 und 3 existieren. Für Fehler der niedrigeren Klassen ist im Zweifelsfall der Kunde beizuziehen um die Details und den Schweregrad zu besprechen.

Fehlerklasse	Kritikalität	Beschreibung
1	Minor	Schönheitsfehler, keine Einbussen für den Anwender
2	Major	Schwere Fehler, Software läuft weiter jedoch mit Interaktion
3	Critical	Kritische Fehler, z.B. Systemabsturz, Freeze

4 Vorgehensweisen

4.1 Unit, Unit-Integration

Für Unit und Unit-Integrationstests wird im White-Box Verfahren gearbeitet. Die Entwickler kennen das erwartete Verhalten und können entsprechende Tests formulieren. Weiter sind sie angehalten sich dabei im Test-First zu üben

4.2 Integrationstests

Für die Verbindungen der einzelnen Komponenten werden Positiv- wie auch Negativtests definiert, die valide mögliche Szenarien bei der Anwendung der Software Abdecken.

4.3 Systemtests

Die Systemtests können im klassischen Black Box Verfahren durchgeführt werden. Das erwartete Resultat wird für definierte Abläufe formuliert. Das live System wird dagegen geprüft.

4.4 NFA-Tests

Um die nicht funktionalen Anforderungen zu Testen werden möglicherweise Anpassungen am System selber nötig um gewisse Szenarien zu provozieren. Auch wird von weiter aussen her Einfluss auf das System genommen um beispielsweise einen Netzausfall zu simulieren.

5 Anforderungen

Nachfolgend die aus den Dokumenten des Kunden gewonnenen Anforderungen

- Die Logger Komponente soll einfach in bestehende Java Applikationen einzubinden sein.
- Durch einfache Methodenaufrufe über die Logger Komponente können Textbasierte Nachrichten Dauerhaft gespeichert werden.
- Die Speicherung der geschickten Logs findet zentral auf einem Logger-Server in einem wohl definierten Format statt.

Auszug aus den Anforderungen des Kunden.

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[1]. Damit kann definiert werden, welche Meldungen (mit welchem Level) tatsächlich übertragen werden. Dieser Level kann zur Laufzeit geändert werden.[2]
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.

[1] Nicht nachträglich von aussen steuerbar, sondern einmalig beim Starten festgelegt.

[2] Nach Rückfrage beim Auftraggeber ist keine Konkretisierung zu der Anforderung gekommen. Diese wird vorläufig als nichtig betrachtet.

6 Testfälle

Die Testfälle / deren ID's beziehen sich direkt auf die Nummer der entsprechenden Anforderung. Es können auch mehrere Testfälle zu einer Anforderung entstehen.

6.1 Logger-Komponente über Interface realisieren

6.1.1 Implementation

Testfall ID	1.1
Anforderung	Die Logger-Komponente muss als Komponente (mit Interfaces) realisiert werden.
Art des Checks	Diese Anforderung ist direkt im Code zu prüfen.
Step 1	Prüfen, ob der Logger über das Interface realisiert wurde
Step 2	Prüfen, ob der LoggerSetup über das Interface realisiert wurde

6.2 LogLevel setzen und filtern

6.2.1 Log Level setzen

Testfall ID	**2.1**
Anforderung	Bei jedem Log-Eintrag hat die aufrufende Applikation einen Message-Level mitzugeben.
Art des Checks	**Diese Anforderung ist direkt im Code und als Blackbox zu prüfen.**
Code	
Step 1	Prüfen ob jedes Senden von Logs ebenfalls einen Level enthält.
Step 2	Prüfen ob das Übertragen der Logs den Level mitschickt
Blackbox	
Step 1	Konfiguration des Loggers so anpassen, dass alles geloggt wird (DEBUG)
Step 2	Server und Applikation starten. Wie wild in der Applikation herumklicken und viele Logs generieren / oder bewusst unterschiedliche Aktionen durchführen und so möglichst unterschiedliche Logs generieren
Step 3	Im Persistierten File der Serverkomponente prüfen, ob jeder Logeintrag ein entsprechendes Level enthält

6.2.2 Log Level filtern

Testfall ID	2.2
Anforderung	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.
Art des Checks	Diese Anforderung ist als Blackbox zu prüfen.
Step 1	Konfiguration des Loggers so anpassen, dass alles geloggt wird (DEBUG)
Step 2	Server und Applikation starten. Logs generieren
Step 3	Sicherstellen, dass Logs aller Log Level auf dem File im Server geschrieben werden
Step 4	Schritte 1-3 für die anderen LogLevel wiederholen und sicherstellen, dass jeweils nur die Logs geschrieben werden deren Level unter, oder gleich dem Definierten Level sind.

6.3 Implementation der Logger-Komponenten

6.3.1 Implementation Logger

Testfall ID	**3.1**
Anforderung	Logger: Message erzeugen und eintragen. Eine Applikation kann via Methodenaufruf Messages (Textstrings) loggen.
Art des Checks	**Diese Anforderung ist direkt im Code zu prüfen.**
Step 1	Code review, Implementation prüfen ob String geloggt werden können

6.3.2 Implementation LoggerSetup

Testfall ID	**3.2**
Anforderung	LoggerSetup: Dient zur Konfiguration des Message Loggers.
Art des Checks	**Diese Anforderung ist direkt im Code zu prüfen.**
Step 1	Code review, Implementation prüfen ob LoggerSetup den eigentlichen Logger konfiguriert

6.4 Aufzeichnung der Logs

6.4.1 Normales Szenario

Testfall ID	**4.1**
Anforderung	Die Log-Ereignisse werden durch Logger-Komponente und den Logger-Server kausal und verlässlich aufgezeichnet.
Art des Checks	**Diese Anforderung ist als Blackbox zu prüfen.**
Step 1	Mit laufendem Server und gestarteter Applikation ist zu prüfen, ob die Logs sauber persistiert werden
!	Gegebenenfalls muss dazu der Code der Applikation angepasst werden. > Einen Counter integrieren, so dass sichergestellt werden kann, dass alle Logs vom Serverteil persistiert wurden

6.4.2 Ausfall Szenario

Testfall ID	**4.2**
Anforderung	Die Log-Ereignisse werden durch Logger-Komponente und den Logger-Server kausal und verlässlich aufgezeichnet.
Art des Checks	**Diese Anforderung ist als Blackbox zu prüfen.**
Step 1	Mit laufendem Server und gestarteter Applikation ist zu prüfen, ob die Logs sauber persistiert werden
Step 2	Die Netzwerk Verbindung zwischen Client und Server wird unterbrochen
Step 3	Es werden Logs generiert
Step 4	Die Verbindung wird wieder hergestellt
Step 5	Alle generierten Logs (auch während des Netzausfalles) werden an den Server übermittelt und persistiert.

6.5 Austauschbarkeit der Logger- Komponenten

6.5.1 Anderen Logger einbinden

Testfall ID	**5.1**
Anforderung	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.
Art des Checks	**Diese Anforderung ist als Blackbox zu prüfen.**
Beschreibung	Die ganze Logger Komponente muss als runnable Jar erstellt werden, so dass diese ohne IDE gestartet werden kann. Durch die eingelesene Konfiguration der LoggerSetupFactory kann das Logger Jar angegeben werden welches beim Starten der Applikation verwendet werden soll.
Step 1	Jar der Logger-Komponente eines anderen Teams im entsprechenden Ressourcenverzeichnis der Applikation hinterlegen,
Step 2	LoggerSetupFactory Konfiguration so anpassen, dass diese das neu hinterlegte Jar anzieht
Step 3	Applikation starten und prüfen ob der eingebundene Logger funktioniert.

6.6 Mehrere Logger parallel

6.6.1 Starten mehrerer Logger auf einem Client

Testfall ID	**6.1**
Anforderung	Es muss möglich sein, dass mehrere Instanzen der Logger-Komponente parallel auf den zentralen Logger-Server loggen.
Art des Checks	**Diese Anforderung ist als Blackbox zu prüfen.**
Step 1	Auf einem Computer wird die Applikation mehrfach mit derselben Konfig gestartet.
Step 2	Prüfen ob die Logs aller Applikationen auf dem Server persistiert sind und voneinander unterschieden werden können

6.6.2 Starten je eines Loggers auf mehreren Clients

Testfall ID	**6.2**
Anforderung	Es muss möglich sein, dass mehrere Instanzen der Logger-Komponente parallel auf den zentralen Logger-Server loggen.
Art des Checks	**Diese Anforderung ist als Blackbox zu prüfen.**
Step 1	Auf mehreren Computern wird die Applikation jeweils einmal gestartet
Step 2	Prüfen ob die Logs aller Applikationen auf dem Server persistiert sind und voneinander unterschieden werden können

6.7 Speicherung der Daten

6.7.1 Lesbarkeit des Logfiles auf dem Server

Testfall ID	**7.1**
Anforderung	Die dauerhafte Speicherung der Messages erfolgt auf dem Server in einem einfachen, lesbaren Textfile.
Art des Checks	**Diese Anforderung ist als Blackbox zu prüfen.**
Step 1	Das Textfile auf dem Server wird geöffnet und soll einfach lesbar sein (nicht binär)

6.7.2 Inhalt der Logs

Testfall ID	**7.2**
Anforderung	Das Textfile enthält mindestens die Quelle der Logmeldung, zwei Zeitstempel (Erstellung Message, Eingang Server), den Message-Level und den Message-Text.
Art des Checks	**Diese Anforderung ist als Blackbox zu prüfen.**
Step 1	Das Textfile auf dem Server wird geöffnet
Step 2	Die Quelle der Meldung ist vorhanden
Step 3	Ein Zeitstempel zum Zeitpunkt der Erstellung des Logs ist vorhanden
Step 4	Ein Zeitstempel zum Zeitpunkt des Empfangs auf dem Server ist vorhanden
Step 5	Der Log Level ist vorhanden
Step 6	Der Message Text ist vorhanden

6.8 StringPersistor Implementation

6.8.1 Lesbarkeit des Logfiles auf dem Server

Testfall ID	**8.1**
Anforderung	Für das Schreiben des Textfiles auf dem Server ist die vorgegebene Schnittstelle StringPersistor zu verwenden und auch dafür eine passende Komponente (StringPersistorFile) zu implementieren.
Art des Checks	**Diese Anforderung ist im Code zu prüfen.**
Beschreibung	Das Interface StringPersistor wird von der API zur Verfügung gestellt und von StringPersistorFile implementiert. Die überschriebenen Methoden werden dabei mit mehreren Unit-Tests getestet. Der StringPersistorFile arbeitet intern mit Instanzen von der Klasse PersistedString. Für den Austausch von Messages vom Client zum Server 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 verschiedenen Unit-Tests getestet, indem ein log im jeweiligen File gespeichert wird und dieser mittels assert-Methoden identisch mit dem abgesendeten ist. Für I/O Aufgaben wie anlegen des Verzeichnisses oder auch Schreib-/Leseberechtigungen auf dem File gibt es weitere Testfälle.

6.9 Adapter Pattern auf StringPersistor

6.9.1 Lesbarkeit des Logfiles auf dem Server

Testfall ID	**9.1**
Anforderung	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.
Art des Checks	**Diese Anforderung ist im Code zu prüfen.**
Beschreibung	Es ist via Codereview zu sicher zu stellen, dass das GoF Pattern umgesetzt wurde. Auch ist die vorgeschriebene Schnittstelle implementiert.