Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Verteilte Systeme und Komponenten
I.BA_VSK.F18

Projektauftrag: Message-Logger

Version 1.0.0	12.02.2018	Doz. Team	1. Fassung

R. Diehl, M. Jud, R. Gisler

Inhalt

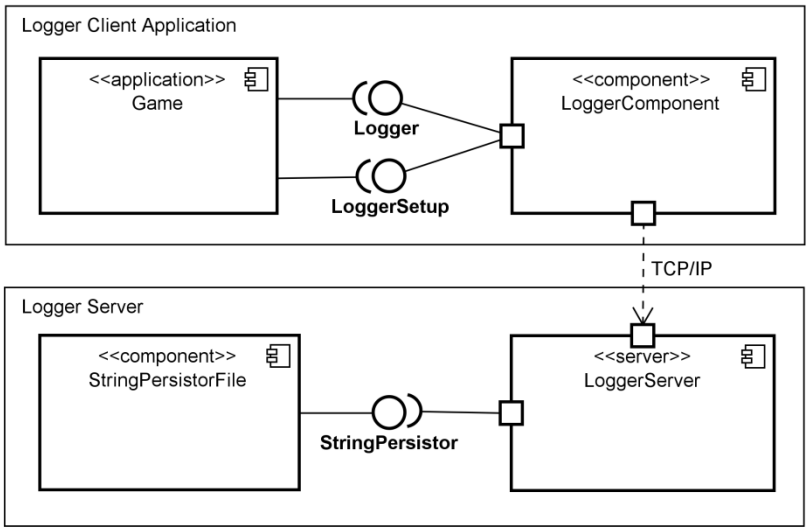
1. Einleitung

Verschiedene Betriebssysteme stellen integrierte Logging-Mechanismen zur Verfügung: Unix-basierende Systeme z.B. das so genannte 'Syslog' und Windows-basierende Systeme das 'Event-Log'. Im Rahmen dieser Aufgabe soll ein alternatives und komponentenbasierendes Message-Loggingsystem entwickelt werden, welches unabhängig von der Plattform und auch auf mehrere Hosts verteilt verwendet werden kann.

2. Anforderungen

2.1. Grundfunktionalität des Message-Loggers

Bei der **Logger-Komponente** handelt es sich um eine Software-Komponente, die sehr einfach in bestehende Java-Applikationen eingebunden werden kann. Eine Applikation kann durch einfache Methodenaufrufe textbasierende Meldungen (Messages) über die Logger-Komponente dauerhaft aufzeichnen. Die Meldungen aller Applikationen sollen in einem zentralen Logger Server in einem wohldefinierten Format gespeichert werden.



2.2. Einsatzgebiet

Das Message-Loggingsystem kann im Allgemeinen z.B. für Debug-Zwecke (Tracing, die Applikation meldet durch Meldungseinträge den zurückgelegten Programmfluss und Zwischenresultate) oder für das Error-Logging im Feld (die Applikation meldet wichtige Fehlersituationen und notwendige Information dazu) verwendet werden.

Im Rahmen des VSK-Moduls soll die Logger-Komponente in eine bestehende Applikation integriert werden. Überlegen Sie sich dafür sinnvolle Ereignisse und Situationen, welche Sie loggen wollen und integrieren Sie die entsprechenden Aufrufe.

Achten Sie darauf, dass Sie die verschiedenen verlangten Loglevels sinnvoll und konsistent nutzen. Da das Spiel auch über Netzwerk gespielt werden kann und somit mehrere Clients und ggf. auch ein Server vorhanden ist, stellt sich auch die Frage wie die Messages miteinander in Korrelation gebracht werden können!

2.3. Muss-Features bei Zwischenabgabe

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 Unittests. Die vorgegebene Schnittstelle **StringPersistor** muss eingehalten werden.

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 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 Netzwerkunterbrüchen umgehen kann.
14	Implementation eines Viewers in irgendeiner Form…

2.5. Realisierung

- Die Planung muss für ein Vorgehen gemäss dem im Modul Projektmanagement Basic vermittelten SoDa mit vier Sprints aufgesetzt werden. Es sind zwei Sprints bis zur Zwischenabgabe SW08 (gemäss Agenda) und zwei Sprints bis zur Schlussabgabe SW13 durchzuführen.
- Dieses Projekt muss vollständig in Java mit NetBeans[1] entwickelt werden. Es muss in den vorgegebenen Projekten auf GitLab verwaltet und auf dem Buildserver laufend integriert werden (Informationen zur dieser Infrastruktur folgen in SW03). Es geht darum, praktische Erfahrung bei Projektmanagement, Spezifikation, Design, Implementierung, Test, Dokumentation und Deployment eines komponentenbasierten und verteilten Systems zu sammeln.
- Spezifikation und Design sind mittels der im Modul Projektmanagement Basic (UML Kurzreferenz, Oestereich) vermittelten UML-Elemente zu beschreiben.
- Die Implementierung soll nur auf den Java Standard-Bibliotheken aufbauen.
- Die Implementation von Unittests wird nicht nur für Feature 9 und 11 verlangt, sondern generell (wo immer möglich) empfohlen, um die Testabdeckung zu erhöhen.
- Achten Sie auf die Codequalität und versuchen Sie die Prinzipien und Praktiken von Clean Code [Developer] anzuwenden.

3. Vorgehen

3.1. Lernziele für das Projektteam

- Sie können ein Softwareprojekt in kleinen Gruppen gemäss *SoDa* in Sprints durchführen. Dabei können Sie die Arbeitsteilung geeignet organisieren.
- Sie kennen den Wert klar definierter Anforderungen.
- Sie kennen die Vorteile von Komponenten.
- Sie können in einem Projekt mit beschränkten Ressourcen umgehen und ein Controlling anwenden. Sie können sich im Spannungsfeld „Dauer-Umfang-Qualität-Kosten (= aufgewendete Zeit)“ bewegen.
- Sie schätzen und kontrollieren Ihre Aufwendungen für das Projekt.
- Sie können komponentenbasierte Architekturen und das Design von Komponenten mithilfe der UML angemessen und nachvollziehbar dokumentieren.
- Sie arbeiten mit zeitgemässen Werkzeugen und Methoden.
- Sie können eine Review organisieren und abwickeln.
- Sie kennen Verfahren, Werkzeuge und Bedeutung des Testens (insbesondere Unit- und Systemtest) und können diese Tests angemessen planen und Ergebnisse nachvollziehbar dokumentieren.
- Sie erstellen Unittests und entwickeln exemplarisch nach dem Testfirst-Verfahren.
- Sie sammeln Erfahrung mit ‚Continuous Integration‘ (CI) durch die konsequente Nutzung von Versionskontrollsystem, automatisiertem Buildprozess, statischer Codeanalyse und dem Buildserver.
- **Jedes Teammitglied kennt auch die Arbeiten der anderen Teammitglieder.**

[1] oder eine andere IDE

3.2. Organisation der Gruppen

Es sind **4er Teams** zu bilden.

3.3. Wichtige Punkte

3.3.1. Vorprojekt

Die Dozierenden geben das Spiel Game-of-Life als Java Code ab, das im VSK-Projekt als Applikation verwendet werden soll. Die Gruppen analysieren die Applikation und bestimmen, welche Ereignisse und Situationen zu loggen sind.

3.3.2. Initialisierungsphase

In der Initialisierungsphase des VSK-Loggerprojektes sind folgende Dinge zu beachten:

Interface Komitee
Die Gruppen bestimmen beim Projektstart je eine/n Delegierte/n in die Interface-Komitees.
Es werden drei unabhängige Komitees gebildet, die je ein Set von Interfaces für die Logger-Komponente definieren und dokumentieren (analog zum vorgegebenen StringPersistor Interface).
Das Interface ist so zu gestalten, dass die entstehenden Logger-Komponenten **interoperabel** sind, d.h. **alle** Gruppen müssen dieses Interface verwenden.

Die Vorschläge werden dem Plenum präsentiert und es wird gemeinsam in Absprache mit dem Dozierendenteam entschieden, welches Set von Interfaces dann von allen Teams realisiert wird.
Die Komitees konstituieren sich selbst, d.h. sie bestimmen den Vorsitz des Interface Komitees sowie die Verantwortung für die Organisation, Zeitplanung, Dokumentation und Änderungsdokumentation (sofern notwendig).

Entwicklungsumgebung
Die Teams setzen ihre Rechner so auf, dass sie mit der vorgegebenen Infrastruktur (Projekt-Template, Versionskontrollsystem und CI-Umgebung, wird ab SW03 eingeführt) funktionieren.
Das Spiel wird ebenfalls in diese Infrastruktur integriert.

ProductBacklog und SprintBacklog für Sprint 1
Aus den Anforderungen in dieser Aufgabenstellung ist das Produktbacklog für den VSK-Logger zu erarbeiten. Bestimmen Sie die Stories bzw. Epics, welche im ersten Sprint umgesetzt werden sollen und schätzen Sie den Aufwand für deren Umsetzung (Design, Implementierung und Test).
Überprüfen Sie Ihre Schätzungen indem Sie Ihre tatsächlichen Aufwendungen aufzeigen (Projektcontrolling).

3.3.3. Meilenstein-Review

Die Teams reviewen sich gegenseitig im Hinblick auf die Schlussabgabe von SW13. Die Partner-Gruppe wird durch die Dozierenden bestimmt.

Die Partner-Teams überprüfen, ob die Artefakte (Dokumentation, Code, Tests etc.) den Anforderungen für die Schlussabgabe fachlich genügen (gemäss Vorgaben aus Modul IM) und vollständig sind. Die Teams erstellen einen Review-Bericht, dieser Bericht wird der Partner-Gruppe ausgehändigt und ist Teil von deren Zwischenabgabe.

Für die Organisation des Reviews sind die Gruppen selber verantwortlich. Jede Gruppe achtet darauf, dass für die eigene Arbeit rechtzeitig, d.h. spätestens SW12 ein Review-Bericht vorliegt.

3.3.4. Präsentationen

Die eigene Lösung wird bei der Schlussabgabe in einer Präsentation vorgestellt.
Die Schwerpunkte der Präsentation und der genaue Zeitpunkt werden in Absprache mit den Dozierenden festgelegt.

Interface Besprechung & Freigabe (SW03)

Anwesenheitspflicht für alle eingeschriebenen Studierenden

- Präsentation der Interfaces der drei Interface-Komitees
- Diskussion der Interfaces im Plemum
- Abstimmung im Plemum über das zu implementierende Interface – Freigabe

Zwischenabgabe (SW08)

Anwesenheitspflicht für alle eingeschriebenen Studierenden

- Demo des Message-Loggersystems, Release 1
- Vollständige Dokumentation (.pdf), Release 1
- Ausserhalb der IDE lauffähige Programme welche:
  - in den vorgegebenen Projekten auf GitLab verwaltet werden
  - auf dem Buildserver laufend integriert wurden
  - automatisierte Tests ohne Fehler durchlaufen
  - eine begründete minimale Codeabdeckung erfüllen

Schlussabgabe (SW13)

Anwesenheitspflicht für alle eingeschriebenen Studierenden

- Präsentation der eigenen Lösung
- Demo der Software mit eigener Loggerkomponente
- Demo der Integration einer fremden Loggerkomponente (SW14)
- Review durchgeführt und protokolliert
- Vollständige Dokumentation (.pdf und ausgedruckt), Release 1 und 2
- Ausserhalb der IDE lauffähige Programme (analog zur Zwischenabgabe!)

3.4. Durchführung der Arbeit

Es stehen für die Arbeit ca. 1/5 der Unterrichtszeit des Modul VSK zur Verfügung. Der Rest der Arbeit muss in der Selbststudienzeit organisiert und durchgeführt werden!

4. Organisation

4.1. Vorgehen und Termine

Vorprojekt, Inititalisierung SW01 bis SW03, Projektlaufzeit: SW04 bis SW14

Die folgenden **Termine sind einzuhalten, die geforderten Arbeitsergebnisse sind Teil der Testatbedingung** (vollständige Teampräsenz für die Besprechung mit dem Dozenten, Termineinhaltung). Es ist den Teams überlassen zusätzliche und „schärfere" Termine zu planen.
Hinweis: Alle Grundlagen für die Besprechungen mit den Dozierenden müssen *schriftlich* vorliegen.

	Inhalte der Besprechung
SW02	Spiel analysiert, *Analyse in ILIAS Briefkasten.*
SW03	Von den Interface-Teams liegen erste dokumentierte Versionen des Interfaces vor. => Mo. 5. März 2018, 21:00 ILIAS Briefkasten *Präsentation und Besprechung im Plenum* Di. 6. März 2018, *Freigabe mit den Dozenten.*
SW03	Organisation der Gruppe ist definiert (SoDa-Rollen); erste Risikoliste Produktbacklog und Sprintplanung für Sprint 1 liegen vor und sind im PMP dokumentiert. *Kurzbesprechung mit dem Dozenten, Protokoll durch Gruppe*
SW05	Dokumentationsplan. Liste der Konfigurations-Items. Spezifikation der drei Elemente für das Systemtesting einschliesslich der Definition des Vorgehens liegt vor. Entwicklung Sprint 1 abgeschlossen. Code wird in GitLab verwaltet und laufend integriert. Erste (geforderte) Unit-Tests laufen erfolgreich. Sprint 2 geplant (SprintBacklog). *Alle Projekte sind auf dem Buildserver fehlerfrei buildbar, es liegen Testfälle vor, Kontrolle durch Dozent*
SW08	Demonstration / Präsentation (Zwischenabgabe Mo. 9./Di. 10. April 2018) Sprint 2 abgeschlossen. Architektur ist festgelegt und exemplarisch dokumentiert. Release 1 ist lauffähig und kann demonstriert werden. Sprint 3 ist geplant (SprintBacklog). Vorgängig Abgabe der Dokumentation & Projektcontrolling (elektronisch) => So. 8. April 2018, 21:00 ILIAS Briefkasten, Peer Review ist organisiert (personell und zeitlich). *Kurzbesprechung mit dem Dozenten, Protokoll durch Gruppe*
SW13	Sprint 4 abgeschlossen. Nachgeführte Softwarespezifikation liegt vor und ist reviewed. Alle Komponenten sind lauffähig und können demonstriert werden. Die Interoperabilität der Logger-Komponente kann demonstriert werden. Demonstration / Präsentation (Schlussabgabe Mo. 14./Di. 15. Mai 2018) Vorgängig Abgabe der Dokumentation (elektronisch + Papier). => So. 13. Mai 2018, 21:00 ILIAS Briefkasten, *Kurzbesprechung mit dem Dozenten, Protokoll durch Gruppe*

5. Beurteilung der Arbeit

Die Durchführung und die Abgabe der Arbeit ist Testatbedingung im Modul VSK. In Ausnahmefällen (ungenügende Mitarbeit in der Gruppe, kein ersichtlicher individueller Beitrag, fehlende Präsenz bei Zwischen- bzw. Schlussabgabe) kann das Testat auch einem einzelnen Gruppenmitglied verweigert werden.