

Projektdokumentation

SOFTWAREARCHITEKT: PHILIPP RIMMELE – PHILIPP.RIMMELE@STUD.HTWK-LEIPZIG.DE

PRODUCT OWNER: ANNA HEINRICH – ANNA.HEINRICH@STUD.HTWK-LEIPZIG.DE

MARIAN GEISSLER – MARIAN.GEISSLER@STUD.HTWK-LEIPZIG.DE

HTWK Leipzig

Inhaltsverzeichnis

I	Anforderungsspezifikation	3
I.1	Initiale Kundenvorgaben	3
I.2	Produktvision	3
I.3	Liste der funktionalen Anforderungen	3
I.4	Liste der nicht-funktionalen Anforderungen	3
I.5	Weitere Zuarbeiten zum Produktvisions-Workshop	3
I.6	Zuarbeit von Autor X	3
I.7	Zuarbeit von Autor Y	3
I.8	Liste der Kundengespräche mit Ergebnissen	4
II	Architektur und Entwurf	4
II.1	Zuarbeiten der Teammitglieder	4
II.2	Entscheidungen des Technologieworkshops	4
II.3	Überblick über Architektur	4
II.4	Definierte Schnittstellen	4
II.5	Liste der Architekturentscheidungen	4
III	Prozess- und Implementationsvorgaben	4
III.1	Definition of Done	4
III.2	Coding Style	4
III.3	Zu nutzende Werkzeuge	5
IV	Sprint 1	5
IV.1	Ziel des Sprints	5
IV.2	User-Stories des Sprint-Backlogs	5
IV.3	Liste der durchgeführten Meetings	6
IV.4	Ergebnisse des Planning-Meetings	6
IV.5	Aufgewendete Arbeitszeit pro Person+Arbeitspaket	6
IV.6	Konkrete Code-Qualität im Sprint	6
IV.7	Konkrete Test-Überdeckung im Sprint	6
IV.8	Ergebnisse des Reviews	6
IV.9	Ergebnisse der Retrospektive	6
IV.10	Abschließende Einschätzung des Product-Owners	6
IV.11	Abschließende Einschätzung des Software-Architekten	6
IV.12	Abschließende Einschätzung des Team-Managers	6
V	Sprint 2	6
VI	Dokumentation	6
VI.1	Handbuch	6
VI.2	Installationsanleitung	7
VI.3	Software-Lizenz	7
VII	Projektabschluss	7
VII.1	Protokoll der Abnahme und Inbetriebnahme beim Kunden	7

VII.2	Präsentation auf der Messe	7
VII.3	Abschließende Einschätzung durch Product-Owner	7
VII.4	Abschließende Einschätzung durch Software-Architekt	7
VII.5	Abschließende Einschätzung durch Team-Manager	7

I. ANFORDERUNGSSPEZIFIKATION

I.1 Initiale Kundenvorgaben

Maecenas sed ultricies felis. Sed imperdiet dictum arcu a egestas. In sapien ante, ultricies quis pellentesque ut, fringilla id sem. Proin justo libero, dapibus consequat auctor at, euismod et erat. Sed ut ipsum erat, iaculis vehicula lorem. Cras non dolor id libero blandit ornare. Pellentesque luctus fermentum eros ut posuere. Suspendisse rutrum suscipit massa sit amet molestie. Donec suscipit lacinia diam, eu posuere libero rutrum sed. Nam blandit lorem sit amet dolor vestibulum in lacinia purus varius. Ut tortor massa, rhoncus ut auctor eget, vestibulum ut justo.

I.2 Produktvision

Quisque vel arcu eget sapien euismod tristique rhoncus eu mauris. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Cras ligula lacus, dictum id scelerisque nec, venenatis vitae magna. Cras tristique porta elit, non tincidunt ligula placerat lobortis. Pellentesque quam enim, mattis in cursus eu, blandit et massa. Mauris aliquet turpis blandit elit vehicula sed posuere lectus facilisis. Donec blandit adipiscing tortor, quis lobortis purus eleifend vel. Nam a tellus at magna scelerisque blandit vel nec erat.

I.3 Liste der funktionalen Anforderungen

XXX

I.4 Liste der nicht-funktionalen Anforderungen

XXX

I.5 Weitere Zuarbeiten zum Produktvisions-Workshop

XXX

I.6 Zuarbeit von Autor X

XXX

I.7 Zuarbeit von Autor Y

XXX

I.8 Liste der Kundengespräche mit Ergebnissen

Datum	Anliegen oder Fragen	Ergebnisse
02.11.18	Wie genau soll das Layout des Diagramms anpassbar sein?	Das Layout soll sowohl manuell als auch automatisch optimiert werden können.
	Reicht es für den ersten Sprint, wenn PUML als Kommandozeilenprogramm umgesetzt wird?	Es soll möglichst früh eine grafische Oberfläche entwickelt werden. Deren Funktionsumfang darf zu Beginn ruhig minimal sein. Wichtig ist, dass das Team möglichst früh einen „optischen Erfolg“ zu verzeichnen hat.

II. ARCHITEKTUR UND ENTWURF

II.1 Zuarbeiten der Teammitglieder

II.1.1 Commandline Funktionalität

Eine der Anforderungen an die Software ist die Bedienung des Programms mittels Kommandozeile. Folgende zwei Möglichkeiten scheinen für die Verwendung im Projekt PUML als sinnvoll. Zum einen ist die Parameterabfrage über eine eigene Implementation auf Basis der Hauptklasse möglich mittels `public static void main(String [] args)`, zum Anderen ist die Nutzung der „Commons CLI“-Bibliothek von Apache eine Option.

Während der Recherche zeigte sich, dass die Nutzung der Commons CLI - Bibliothek sehr gut dokumentiert ist und in der Praxis oft Anwendung findet, auch das Umsetzen eines Tests schien weniger problematisch zu gelingen, als ein Abfragen der Parameter über `String [] args` im Hauptprogramm. Aus diesem Grund wird an dieser Stelle die Apache Bibliothek kurz vorgestellt. Ein Download der Bibliothek erfolgt über die Seite des Entwicklers Apache¹ und muss anschließend in die Entwicklungsumgebung eingebunden, sowie in das Programm importiert werden.

Die Arbeit mit Commons CLI lässt sich grundsätzlich in drei Schritte unterteilen, Parameterdefinition, das Einrichten des Parsers und die Verkettung mit der jeweiligen Funktion.

Zuerst wird festgelegt, welche Parameter der Anwendung übergeben werden, hierzu wird ein neues Container Objekt vom Typ `Options` angelegt. Anschließend werden die gewünschten Befehle mit den entsprechenden Parametern dem Container hinzugefügt, so dass später ein Aufruf im Terminal möglich ist, wie beispielsweise `ls -al meinfile.txt` um die Zugriffsrechte einer Datei zu überprüfen.

```
//Erzeugt neuen Container fuer Programmparameter
Options options = new Options();
//Hinzufuegen einer neuen Option
options.addOption("l",false, "Alle Leerzeichen entfernen.");
```

Zunächst wird ein Parser initialisiert, während anschließend über eine logische Verknüpfung der Flags die entsprechende Funktion aufgerufen wird. Wichtig ist in diesem Zusammenhang noch die Verwendung von Exceptions zu erwähnen, die entweder durch den Ausdruck `ParseException` aus der Bibliothek oder `try / catch` Schlüsselwörter abgefangen werden müssen.

```
CommandLineParser parser = new DefaultParser();
CommandLine commandLine = parser.parse(options,args);
```

¹<http://commons.apache.org/proper/commons-cli/>

```

if(commandLine.hasOption("b"))
{
    System.out.println("String eingebe: ");
    String myString = keyboard.nextLine(); //String einlesen
    getWordBefore('.',myString); // liefere alle Woerter vor Punkt
}
if (commandLine.hasOption("l"))
{
    String myString = "g e s p e r r t g e s c h r i e b e n";
    System.out.println(noSpace(myString)); //entfernt alle Leerzeichen
}

```

Am Ende muss das Programm übersetzt werden und steht anschließend zur Nutzung mit den gesetzten Parametern zur Verfügung. So liefert in diesem Fall die Eingabe im Terminal: `java MeinProgrammName -l` die Ausgabe „gesperrtgeschrieben“ zurück.

II.2 Entscheidungen des Technologieworkshops

XXX

II.3 Überblick über Architektur

XXX

II.4 Definierte Schnittstellen

XXX

II.5 Liste der Architekturentscheidungen

XXX (bewusste und unbewusste Entscheidungen mit zeitlicher Einordnung)

III. PROZESS- UND IMPLEMENTATIONSVORGABEN

III.1 Definition of Done

XXX

III.2 Coding Style

Bitte die Datei `javaCodeStyle.xml` im specification-Verzeichniss in Eclipse importieren und verwenden. Hierfür in Eclipse unter „Window->Preferences->Java->Code Style->Formatter“ auf Import klicken und die XML-Datei auswählen.

Ist der passende Coding Style eingestellt kann der Quellcode mit „STRG+SHIFT+F“ automatisch formatiert werden. Wird dies vor jedem Commit gemacht, entsteht ein einheitlicher Code-Style und die Änderungen können gut mit GIT überprüft werden.

Des weiteren empfiehlt es sich bei größeren oder stark geschachtelten Code-Abschnitten die Zugehörigkeit der Schließenden Klammer mit einem Kommentar zu Kennzeichnen.

Sonstige Konventionen:

- Variablen und Instanzen beginnen kleingeschrieben
- Klassen und Interfaces beginnen mit Großbuchstaben

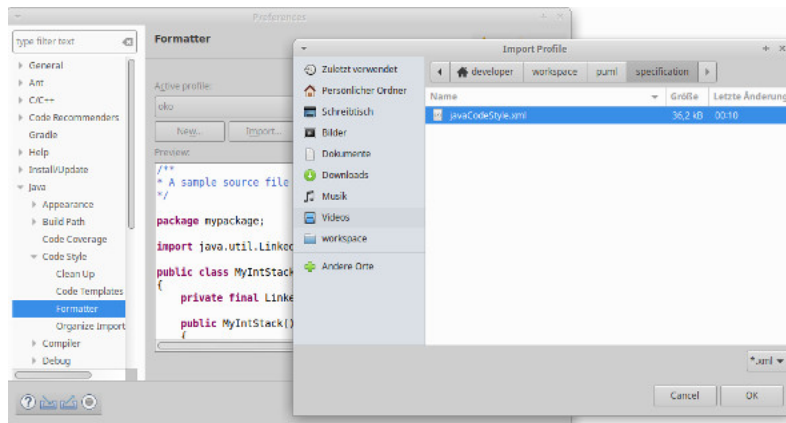


Abbildung 1: Code-Style in Eclipse importieren

- Besteht ein Namen aus mehreren zusammengesetzten Wörtern, beginnen alle weiteren Wörter mit Großbuchstaben (keine Unterstriche in Namen verwenden)
- Aussagekräftige Namen verwenden
- Alle Namen auf Englisch
- Die Kommentare auf Deutsch

III.3 Zu nutzende Werkzeuge

- Eclipse - Entwicklungsumgebung
- GIT - Dateiversionierung
- Meld - Unterschiede zwischen Dateien anzeigen
- Texmaker - Latex-Editor
- GIMP - Bildbearbeitung für das Editieren von Screenshots

IV. SPRINT 1

IV.1 Ziel des Sprints

XXX

IV.2 User-Stories des Sprint-Backlogs

XXX

IV.3 Liste der durchgeführten Meetings

XXX

IV.4 Ergebnisse des Planning-Meetings

XXX

IV.5 Aufgewendete Arbeitszeit pro Person+Arbeitspaket

Arbeitspaket	Person	Start	Ende	h	Artefakt
Dummyklassen	Musterstudi	3.5.09	12.5.09	14	Klasse.java
AP XYZ					

IV.6 Konkrete Code-Qualität im Sprint

XXX

IV.7 Konkrete Test-Überdeckung im Sprint

XXX

IV.8 Ergebnisse des Reviews

XXX

IV.9 Ergebnisse der Retrospektive

XXX

IV.10 Abschließende Einschätzung des Product-Owners

XXX

IV.11 Abschließende Einschätzung des Software-Architekten

XXX

IV.12 Abschließende Einschätzung des Team-Managers

XXX

V. SPRINT 2

VI. DOKUMENTATION

VI.1 Handbuch

XXX

VI.2 Installationsanleitung

XXX

VI.3 Software-Lizenz

XXX

VII. PROJEKTABSCHLUSS

VII.1 Protokoll der Abnahme und Inbetriebnahme beim Kunden

XXX

VII.2 Präsentation auf der Messe

Poster, Bericht

VII.3 Abschließende Einschätzung durch Product-Owner

XXX

VII.4 Abschließende Einschätzung durch Software-Architekt

XXX

VII.5 Abschließende Einschätzung durch Team-Manager

XXX