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
Patrick Otte – patrick.otte@stud.htwk-leipzig.de
Johann Gerhardt – johann.gerhardt@stud.htwk-leipzig.de
Michael Lux – michael.lux@stud.htwk-leipzig.de
Jan Sollmann – jan.sollmann@stud.htwk-leipzig.de
Julian Uebe – julian.uebe@stud.htwk-leipzig.de
Elisabeth Schuster – elisabeth.schuster@stud.htwk-leipzig.de
Jona Meyer
Leo Rauschke
Tore Arndt – tore.arndt@stud.htwk-leipzig.de
HTWK Leipzig

Inhaltsverzeichnis

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

I.2.1 Kernfunktionalität

- Einlesen einer Jar-Datei oder mehrerer Java Dateien
- Analyse des Java-Source Codes und Identifikation seiner verbundenen Klassen sowie deren Verknüpfungen und Methoden
- Möglichkeit der Ausgabe eines Klassendiagrams oder eines Sequenzdiagramms
- Sequenzdiagramm:
  - Möglichkeit der Ausgabe eines Sequenzdiagramms
  - Möglichkeit Aufrufe von Methoden im Sequenzdiagramm zu blockieren
- Klassendiagramm:
  - Möglichkeiten des Nutzers der Auswahl der Bestandteile in einem Klassendiagramm
  - Möglichkeit der Voransicht des Klassendiagramms
- Überstützung für das Layout:
  - Layout muss automatisch konfigurierbar sein
  - Layout muss die Möglichkeit haben manuell konfiguriert werden zu können
- Beide Diagrammarten sollen als String oder als Textdatei ausgegeben werden können

Aufgaben für den 1. Sprint:

- Erstellen eines ersten GUI's
- Einlesen der Dateien sowie deren Analyse
- Ausgabe des Klassendiagramms

I.2.2 Anwenderprofil

Denkanstöße:

- Wer ist unsere Zielgruppe?
- Worauf legt unsere Zielgruppe besonderen Wert?
- Was für einen Mehrwert bietet unser Produkt der Zielgruppe?

Hier geht es einerseits um eine klare, wenn auch triviale Einordnung, wer unser Produkt später nutzen soll. Andererseits soll erörtert werden, wie das Produkt unserer Zielgruppe das Leben leichter machen kann.

I.2.3 PlantUML-Vorstellung

Denkanstöße:

- Wie beschreibt man Klassen allgemein mit PlantUML?
- Welche Arten von Beziehungen zwischen Klassen unterscheidet PlantUML (was für Arten von Pfeilen gibt es)?
- Gibt es Konstrukte, die mit PlantUML vielleicht nur unzureichend dargestellt werden könnten?
- Gibt es PlantUML-Erweiterungen, die von besonderem Interesse für unser Projekt sind?

Diese Recherche soll einen ersten Einblick in den Aufbau und die Möglichkeiten von PlantUML gewähren.

I.2.4 Konkurrenzprodukte

Denkanstöße:

- Gibt es schon ähnliche Ansätze?
- Wie unterscheiden / gleichen sich diese bezogen auf ihre Funktionalität?
- Welche Nischen gibt es?

Hier sollen sowohl Marktlücken als auch "Best Practicesïdentifiziert werden.

I.2.5 Mögliche Weiterentwicklung

Denkanstöße:

- Wie können wir unser Produkt verbessern, sobald es die definierten Kernfunktionalitäten enthält?
- (Am besten mit der Person erörtern, die zu Konkurrenzprodukten recherchiert) Welche Features bietet die Konkurrenz, die wir nicht für die Kernfunktionalität festgelegt haben?
- Grob geschätzt: Wie nützlich sind diese Features und wie aufwendig wäre deren Implementierung?

Hier wurde z.B. schon eine Implementierung als Eclipse-Plugin und Support für verschiedene Programmiersprachen angesprochen.

I.2.6 GUI-Anforderungen

Denkanstöße:

- Wie könnte das User Interface aussehen?
- Wie können wir für eine gute User Experience sorgen?

Auch, wenn die GUI im ersten Sprint nur in Grundzügen entwickelt wird, sollten wir uns früh Gedanken über deren Aussehen und Funktionalität machen. Dazu gehört, wie der Workflow abläuft und wie wir das GUI unserer Anwendung darauf zuschneiden können. Welche großen roten Knöpfe brauchen wir, welche Funktionen dürfen in dreifach verschachtelten Untermenüs versteckt werden?

I.2.7 Datenmodell

- Welche Daten verarbeiten wir?
  - Quellcode, der vom Anwender gegeben wird
  - Daten sind abhängig vom Nutzer
- Form der Datenaus- und -eingabe
  - Eingabe: Jar-Datei oder mehrere Java-Dateien, die Quellcode enthalten oder Textdateien
  - Ausgabe: Klassendiagramme, Sequenzdiagramme, ... mit den Beziehungen zwischen den einzelnen Klassen
- wichtige Variablen und Parameter
  - Klassenname
  - Klassenattribute
  - Relationen zwischen den einzelnen Klassen (mit Pfeilen dargestellt)
  - Methoden der einzelnen Klassen
  - Eigenschaften wie abstract, private, protected, etc.
- wichtige Klassen aus der Java-Standardbibliothek
  - java.awt - zum Erstellen von User-Interfaces
  - jav.io - zum Einlesen von Dateien
  - java.util - enthält z.B. event model, frameworks, internationalization, ...
  - javax.swing - enthält Klassen zum Erstellen einer GUI

I.2.8 Erste User-Story

Denkanstöße:

- Stell dir vor, das Produkt ist gerade fertig entwickelt worden und deine Teamleiter wollen ein Plant-UML von deinem Quellcode sehen (PUMLception). Welchen Workflow erwartest du? Was machst du in welcher Reihenfolge?
- Optional: Besprich dich mit dem / der Recherchierenden für die GUI-Anforderungen. Habt ihr unterschiedliche Vorstellungen vom Workflow und wenn ja, wie unterscheiden sie sich?
- Wie würde für dich rein intuitiv das Graphical User Interface aussehen? Wenn du das GUI mit nur drei Knöpfen bauen müsstest, welche Funktionen würdest du ihnen zuweisen?

I.2.9 Workshop-Bedarfsermittlung

Denkanstöße:

- Was für Kompetenzen könnten dem Team für die Entwicklung des Produkts fehlen?

Um die Recherche hier zu vereinfachen, wäre es nicht schlecht, wenn die anderen Recherchierenden den Bearbeitenden auf mögliche Wissenslücken aufmerksam machen. Ziel dieser Recherche ist ein allgemeiner Überblick, wo Defizite vorhanden sind. Es ist zu vermuten, dass die anfänglich identifizierten Probleme eher abstrakter Natur sind - konkrete "Baustellenßeigen sich für gewöhnlich erst in der Entwicklung. Erwartet werden also keine haargenauen Angaben zu Kompetenzen bzw. Inkompetenzen.

I.2.10 Workshop-Recherche

Denkanstöße:

- Wie können wir die Teammitglieder effektiv und effizient auf einen Stand bringen?
- Welche Ressourcen könnten dafür nützlich sein (Scripting-APIs, gute Tutorials etc.)?
- Wie können wir die gefundenen Ressourcen so zur Verfügung stellen, dass jeder einfach darauf zugreifen kann?

Diese Recherche ist einerseits eng mit der Recherche "Workshop-Bedarfsermittlung"verbunden, es schadet also auf jeden Fall nicht, sich über deren Ergebnisse zu informieren. Aber auch unabhängig davon können schon erste Ideen entwickelt werden, wie das Team miteinander und voneinander lernen kann.

Terminabsprache/Mitteilen von Neuigkeiten Zur Terminabsprache eignet sich gut ein Messenger wie zB Telegram, wo das Team eine Gruppe hat, übe die Neuigkeiten und Termine schnell ausgetauscht werden können und sich gebündelt an einem Ort befinden.

Gefundene Ressourcen zur Verfügung stellen Es wäre sinnvoll, gefundene Ressourcen wie zum Beispiel Dokumente über einsetzbare Technologien, Tutorials oder andere Hintergrundinformationen außerhalb vom Gruppenschat teilen zu können, das können wir über git machen.

I.3 Liste der funktionalen Anforderungen

I.3.1 Userstories

Vorschau Als Benutzer wünsche ich mir eine Vorschau der Diagramme, damit ich einschätzen kann ob ich damit zufrieden bin.

Interfaces Als Benutzer wünsche ich mir, dass ich abhängig von Interfaces die zugehörigen Klassen anzeigen lassen kann, damit ich weiß, welche Methoden ich implementieren muss.

Kommandozeile Als Benutzer wünsche ich mir, dass das Programm von der Kommandozeile aus aufrufbar ist, um es automatisiert starten zu können.

Dateien einlesen

Art der eingelesenen Datei Als Benutzer wünsche ich mir, dass eine Auswahl zwischen Jar- und Java-Dateien möglich ist, damit Quellcode nicht doppelt eingelesen wird.

Java-Dateien Als Benutzer wünsche ich mir, dass Java-Dateien einlesbar sind, um den Quellcode von einer oder mehreren Klassen zu analysieren.

Jar-Dateien Als Benutzer wünsche ich mir, dass Jar-Dateien einlesbar sind, um den Quellcode zu analysieren.

Klassendiagramme Als Benutzer wünsche ich mir, Klassendiagramme aus meinem bestehenden Quellcode erstellen zu können, damit ich das nicht manuell tun muss.

Sequenzdiagramme Als Benutzer wünsche ich mir, Sequenzdiagramme aus meinem bestehenden Quellcode erstellen zu können, damit ich das nicht manuell tun muss.

Klassenauswahl Als Benutzer wünsche ich mir die Möglichkeit, Klassen zu selektieren, damit die Diagramme nicht zu unübersichtlich werden.

Methoden- und Variablenauswahl Als Benutzer wünsche ich mir die Möglichkeit, Methoden und Variablen zu selektieren, damit die Diagramme nicht zu unübersichtlich werden.

Multiple Klassenauswahl Als Benutzer wünsche ich mir einen Button, mit dem ich alle Klassen an- oder abwählen kann, damit ich nicht alle Klassen einzeln auswählen muss.

Multiple Methodenauswahl Als Benutzer wünsche ich mir einen Button, mit dem ich alle Methoden an- oder abwählen kann, damit ich nicht alle Methoden einzeln auswählen muss.

Layout Als Benutzer wünsche ich mir, das Layout meiner Diagramme ändern zu können, um deren Aussehen zu verbessern.

Drag-and-Drop Als Benutzer wünsche ich mir, die ausgewählten Dateien oder Ordner per Drag-and-Drop in das Programm aufzunehmen, damit ich den Datei-öffnen-Dialog nicht nutzen muss.

Anzeigen und Speichern von PlantUML Als Benutzer wünsche ich mir, Diagramme als PlantUML-Code anzeigen und speichern zu können, um den Aufbau nachvollziehen zu können.

Speichern von Bilddateien Als Benutzer wünsche ich mir, die erstellten Diagramme als Bilddatei exportieren zu können, um sie in meine Projektdokumentation mit aufzunehmen.

Speichern von Konfigurationen Als Benutzer wünsche ich mir, meine Konfiguration speichern zu können, damit ich meine Präferenz nicht jedes Mal aufs Neue einstellen muss.

Benutzerhandbuch Als Benutzer wünsche ich mir, das Benutzerhandbuch über die GUI anzeigen lassen zu können, damit ich die gedruckte Version nicht benötige.

Übersicht aller Diagramme für ein Projekt Als Benutzer wünsche ich mir eine Übersicht aller Diagramme, die ich für mein Projekt erstellt habe, damit ich leichter auf diese zugreifen kann.

Drucken Als Benutzer wünsche ich mir, Diagramme über das GUI drucken zu können, damit ich die Bilddateien nicht separat öffnen muss.

Exceptions als Sequenzdiagramme Als Benutzer wünsche ich mir, dass der mögliche Pfad der Exceptions als Sequenzdiagramm angezeigt werden kann, um ungehandelte Exceptions zu vermeiden.

I.4 Liste der nicht-funktionalen Anforderungen

I.4.1 Allgemein

- Testabdeckung >= 50%
- Keine spürbare Verzögerungen im Programmablauf
- Bei längeren Ladezeiten muss dies dem Benutzer mitgeteilt werden (Ladebalken)

I.4.2 Userstories

Plattformunabhängigkeit Als Project Owner wünsche ich mir, dass das Programm plattformunabhängig ist, damit es sich gut verbreiten lässt.

Plugin Als Benutzer wünsche ich mir, PUML als Plugin direkt in Eclipse verwenden zu können, damit ich nicht außerhalb meiner Entwicklungsumgebung arbeiten muss.

Installation Als Benutzer wünsche ich mir, dass die Installation unkompliziert ist, damit ich das Programm schnell benutzen kann.

I.5 Weitere Zuarbeiten zum Produktvisions-Workshop

XXX

I.6 Zuarbeit von Autor X

XXX

I.7 Zuarbeit von Autor Y

XXX

I.8 Risikoanalyse

I.9 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.

Warscheinlichkeit	Auswirkung	Gesamt	Risiko	Maßnahmen
5	5	25	Entwickler fällt aus	• Die Aufgaben werden umverteilt • Projektleitung springt ein
3	7	21	Projektleiter fällt aus	Professor Weicker kontaktieren und weitermachen
7	4	28	Die Zeit in einem Sprint reicht nicht	• Krisentreffen • Unterstützung der Verantwortlichen durch andere Entwickler • Sprintziel als nicht erreicht kennzeichnen und in nächsten Sprint übernehmen

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 Enticklers Apache[1] 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 bespielsweise `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 intitialisert, während anschließend über eine logische Verknüpfung der Flags die entsprechende Funktion aufgerufen wird. Wichtig ist in diesem Zusammenhang noch die

[1] http://commons.apache.org/proper/commons-cli/

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);

 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.1.2 GUI

Welche GUI-Frameworks gibt es für Java? Aktuell gibt es folgende Möglichkeiten zur Erstellung einer GUI Oberfläche in Java:

- Swing
- JavaFX
- Standard Widget Toolkit (SWT)
- Abstract Window Toolkit (AWT)
- Google Web Toolkit (GWT)
- Qt (Qt Jambi)
- GTK+

Welche können für das Projekt genutzt werden? Für unser Projekt kommen aktuell die Frameworks Swing und JavaFX in Frage. Zum Einen sind beide aktuell die beiden meist genutzten GUI Frameworks in Java, zum Anderen sind hierfür Eclipse Plugins verfügbar.

Vor- und Nachteile der Frameworks Swing

Vorteile

- Bestandteil des Java Development Kits/Java Foundation Classes
- Nutzt eine Sammlung von Bibliotheken zur GUI-Programmierung (Bspw. AWT)

Nachteile

- Wird nicht mehr weiterentwickelt oder gewartet
- Probleme im Bereich Medieneinbindung und Animation
- Bestimmte Anwendung wie bspw. Zooming nicht möglich

JavaFX Vorteile

- Teil jeder neuen Java SE Installation
- Möglichkeit einfach animierte Übergänge einzubinden
- optisch ansprechender
- Anwendung kann mittels CSS bearbeitet werden (durch Einbindung von FXML-Code)

Nachteile

- weniger Online-Hilfe, kleinere Community

Fazit In Hinsicht auf die Langlebigkeit bzw. der Zukunftssicherheit, der moderneren Optik, der Einbindung in Eclipse und dem steigenden Support haben wir uns für JavaFX zur GUI Entwicklung in unserem Projekt entschieden.

II.1.3 Reguläre Ausdrücke

Reguläre Ausdrücke sind Beschreibungen eines Musters, sog. Patterns, die bei Zeichenkettenverarbeitung eingesetzt werden. Mittels dieser Muster lassen sich Zeichenketten suchen und ersetzen.

Funktionen

(1) Komplette Übereinstimmung suchen

```
Pattern.matches(regex, this);
```

```
Pattern p = Pattern.compile(regex);
Matcher m = p.matcher(input);
return m.matches();
```

(2) Teilstring finden

- alle Vorkommen des Teilstrings innerhalb eines Suchstrings suchen

(3) Teilfolgen ersetzen

(4) Zerlegen einer Zeichenfolge

- Trennzeichen sind durch Muster definiert, resultiert in Sammlung von Zeichenfolgen

Verwendung

Um mit Regulären Ausdrücken arbeiten zu können, wird das Paket ‚java.util.regex' implementiert. Es enthält die Klassen Matcher (Zugriff auf Mustermaschine) und Pattern (Repräsentation RE in vorkompiliertem Format). Außerdem gibt es verschieden Klassifizierungen, um die Suche genauer zu definieren.

Quantifizierung	Anzahl der Wiederholungen
X?	X kommt einmal oder keinmal vor
X*	X kommt keinmal oder beliebig oft vor
X+	X kommt einmal oder beliebig oft vor
X{n}	X muss genau n-mal vorkommen
X{n,}	X kommt mindestens n-mal vor
X{n,m}	X kommt mindestens n-, aber max. m-mal vor

zeichenklasse	Enthält
.	jedes Zeichen
[aei]	Zeichen a, e, i
[^aei]	nicht die Zeichen a, e, i
[0-9a-f]	Zeichen 0-9 oder Kleinbuchstaben a-f
\d	Ziffer: [0-9]
\D	keine Ziffer: [^0-9] bzw. [^\d]
\p{Blank}	Leerzeichen oder Tab: [\t]
\p{Lower}, \p {Upper}	Klein-/Großbuchstaben: [a-z] bzw. [A-Z]

weitere Klassifizierungen:
https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

Beispiele

(1) Rückgabewert beider Abfragen ist true

```
System.out.println(Pattern.matches("'.*'","'Hallo Welt'" ));
System.out.println("'Hallo Welt'".matches("'.*'"));
```

(2) Abfrage nach Teilstring, Rückgabewert ist gefundener Teilstring

```
String text = "Moderne Programmiersprachen haben durch die Vernetzung von Computern
    neue Anforderungen erfahren. So lautet auch ein Motto von Sun: 'The Network is
    the Computer.' ";
Matcher matcher = Pattern.compile("'.*'").matcher(text);
while(matcher.find()) {
   System.out.println(matcher.group());
}
```

II.1.4 XML

XML ist eine Metasprache die genutzt wird um Daten zwischen Anwendungen auszutauschen. Dies wird durch eine hierarchische Struktur realisiert.

Eigenschaften

- im Format einer Textdatei
- von Menschen als auch von Maschinen lesbar
- HTML ist eine Untersprache von XML
- Dokumenttypdefinitionen (DTD) ermöglichen, dass nur bestimmte Strukturen in einem XML-Dokument möglich sind
- ohne DTD gut geeignet für beliebigen Datenaustausch

Vergleich zu JSON (JavaScript Object Notation)

- Vorteile gegenüber JSON:
  - Einfache Lesbarkeit

```xml
<?xml version="1.0" encoding="UTF-8"?>

<Softwareprojekt jahr="2018">
    <Gruppe gID="3">
        <Master>
            <Projektowner mID="1"> Anna
            </Projektowner>
            <Projektleader mID="2"> Oko
            </Projektleader>
        </Master>
        <Bachelor>
            <!-- Hier Mitglieder einfügen -->
        </Bachelor>
    </Gruppe>
</Softwareprojekt>
```

Abbildung 1: Beispielcode XML

  - Etabliertes Austauschformat
  - Erweiterbar

- Nachteile gegenüber JSON:
  - Enthält viel "Balast"der für Datenaustausch nicht nötig ist
  - Datenvolumen relativ hoch
  - Komplexe Syntax

Verwendung

- Erste Zeile im Dokument definiert Version und Codierung
- Struktur in der Form <tag> ... </tag>, wobei <x> das öffnende Tag und </x> das schließende Tag darstellt
- Ein Wurzelknoten wird benötigt, der den gesamten XML-Quelltext umfasst
- Tags sind ineinander geschachtelt
- Attribute können mit Attribut =”Wert” im öffnenden Tag definiert werden

Bibliotheken in Java

- org.xml.sax.* (XML Datei lesen)
- org.w3c.dom.* (einlesen in den Speicher und schreiben in der Datei)
- java.xml.parsers.* (Auslesen der XML-Dateien aus dem Speicher und übernehmbar als DOM-Objekt)

II.1.5 XPath

Abfragesprache zur Addressierung/Auswertung von XML und Grundlage für Standards: XLST, XPointer, XQuery.

Aufbau und Verwendung

- XML-Dokument wird als Baum betrachtet
  - Knoten (nodes): Dokumenten-Knoten, XML-Elemente, -Attribute, -Textknoten, -Kommentare, -Namensräume und -Verarbeitungsanweisungen
  - Achsen: preceding, following, preceding-sibling und following-sibling
- XPath-Ausdruck besteht aus mehreren Lokalisierungsschritten:
  - achse::knotentest[prädikat 1][prädikat 2]...
  - Beispiel: /descendant-or-self::Foo
  - Prädikate: Funktionen/Operatoren zur weiteren Einschränkung
  - Beispiel: text(), comment() für bestimmten Datentyp

Beispiel an dargestellter XML-Datei

- /descendant-or-self::Softwareprojekt bzw. Softwareprojekt

  Wählt alle untergeordnete Knoten inklusive des Kontextknotens Softwareprojekt aus.
- /descendant-or-self::Softwareprojekt/descendant::Gruppe bzw. Softwareprojekt//Gruppe

  Wählt alle untergeordnete Knoten von Gruppe aus.
- /descendant-or-self::Softwareprojekt/descendant::Gruppe/descendant::Master/descendant::Projektowner/attribute oder /Softwareprojekt/Gruppe/Master/Projektowner/attribute::*

  liefert Attribute='mID=1' zurück.

II.2 Entscheidungen des Technologieworkshops

XXX

II.3 Überblick über Architektur

XXX

II.4 Definierte Schnittstellen

XXX

II.5 Liste der Architekturentscheidungen

III. Prozess- und Implementationsvorgaben

III.1 Definition of Done

XXX

Zeit	Entscheidung
Bei Projektvergabe	Als Programmiersprache wird Java verwendet. Begründung der Entscheidung: • Plattformunabhängigkeit • Alle aus dem Team beherrschen Java • Sauber und einsteigerfreundlich • Weiterentwicklung zu Eclipse-Plugin möglich • Ausgereifte GUI-Frameworks verfügbar
Technologieworkshop	Für das GUI wird JavaFX verwendet Begründung der Entscheidung: • ...

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, ensteht 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.

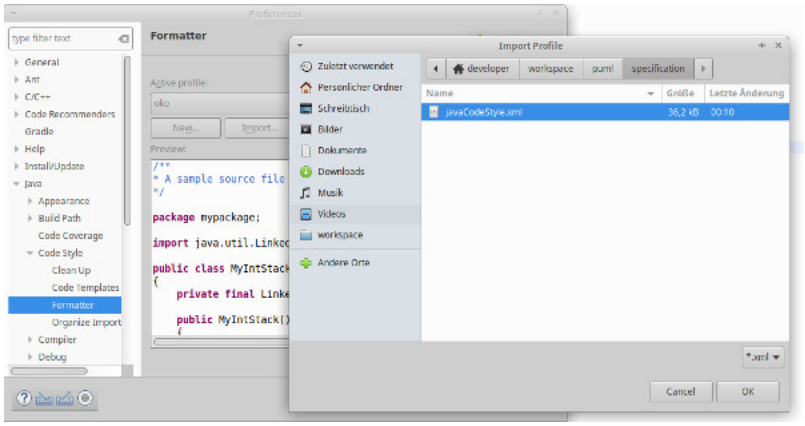

Abbildung 2: Code-Style in Eclipse importieren

Sonstige Konventionen:

- Variablen und Instanzen beginnen kleingeschrieben
- Klassen und Interfaces beginnen mit Großbuchstaben
- 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
- Lange Kommentare immer vor den Codeabschnitt
- Alle Methoden im Javadoc-Stiel dokumentieren

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

Es soll eine funktionsfähige Basisversion, welche für das einfache erstellen von Klassendiagrammen aus Java-Code verwendet werden soll entstehen. Das Programm soll sowohl über die Kommandozeile, als auch über eine grafische Oberfläche bedient werden können. Die erzeugten Klassendiagramme sollen in der grafischen Oberfläche angezeigt werden können.

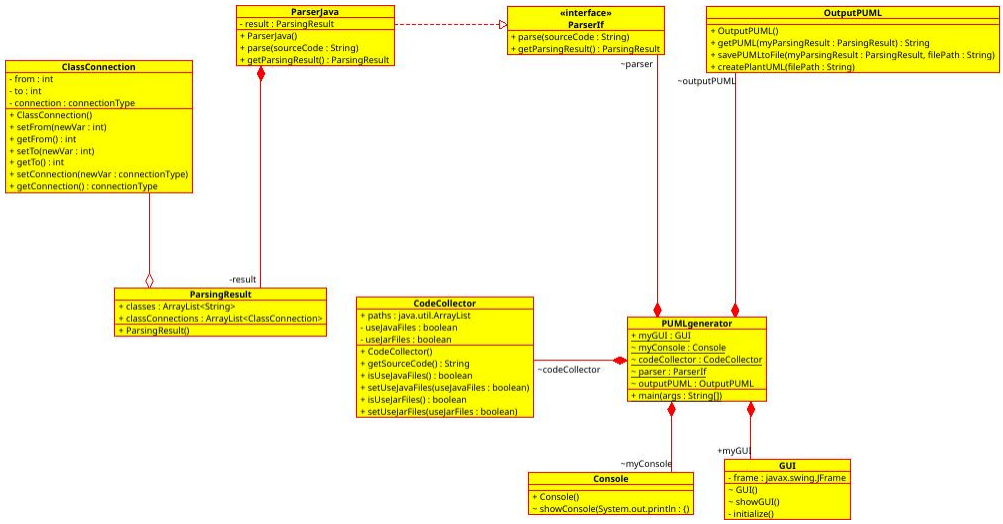

Abbildung 3: Klassendiagramm des Sprints

IV.2 User-Stories des Sprint-Backlogs

IV.2.1 Dateien einlesen

Art der eingelesenen Datei Als Benutzer wünsche ich mir, dass eine Auswahl zwischen Jar- und Java-Dateien möglich ist, damit Quellcode nicht doppelt eingelesen wird.

Java-Dateien Als Benutzer wünsche ich mir, dass Java-Dateien einlesbar sind, um den Quellcode von einer oder mehreren Klassen zu analysieren.

Jar-Dateien Als Benutzer wünsche ich mir, dass Jar-Dateien einlesbar sind, um den Quellcode zu analysieren.

IV.2.2 Vorschau

Als Benutzer wünsche ich mir eine Vorschau der Diagramme, damit ich einschätzen kann ob ich damit zufrieden bin.

IV.2.3 Kommandozeile

Als Benutzer wünsche ich mir, dass das Programm von der Kommandozeile aus aufrufbar ist, um es automatisiert starten zu können.

IV.2.4 Klassendiagramme

Als Benutzer wünsche ich mir, Klassendiagramme aus meinem bestehenden Quellcode erstellen zu können, damit ich das nicht manuell tun muss.

IV.2.5 Anzeigen und Speichern von PlantUML

Als Benutzer wünsche ich mir, Diagramme als PlantUML-Code anzeigen und speichern zu können, um den Aufbau nachvollziehen zu können.

IV.2.6 Plattformunabhängigkeit

Als Project Owner wünsche ich mir, dass das Programm plattformunabhängig ist, damit es sich gut verbreiten lässt.

IV.3 Zeitliche Planung

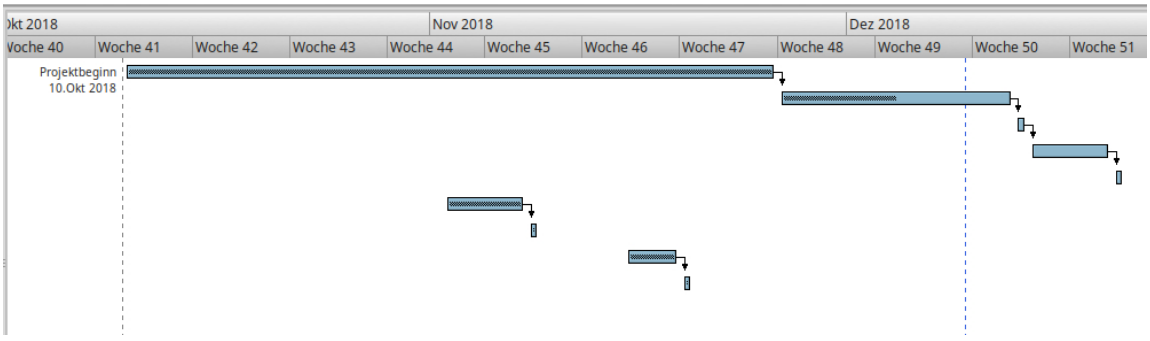

Abbildung 4: Gantt-Diagramm für Sprint 1

IV.4 Liste der durchgeführten Meetings

- Planning-Meeting (29.11.2018)
- Zwischen-Meeting (03.12.2018)
- Review-Meeting (13.12.2018)

IV.5 Ergebnisse des Planning-Meetings

Dem gesammten Team ist die geplante Grundstruktur des Programms bekannt. Jeder weis welchen Teil des Programms er implementieren soll.

IV.6 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.7 Konkrete Code-Qualität im Sprint

XXX

IV.8 Konkrete Test-Überdeckung im Sprint

XXX

IV.9 Ergebnisse des Reviews

Klasse	**Methode**	**Anmerkungen**
Console	showConsole	Pfad anpassen
CodeCollector	-	Unit-Tests für Ordner
CodeCollector	getSourceCode	gleichzeitig .jar- und .java-Dateien
ParserJava	parse	Bug: Entfernt zu viel Source Code! Mehr Tests
OutputPuml	-	generell mehr Kommentare
OutputPuml	getPuml	Redundanter Code mit savePumlToFile, generell mehr Kommentare
OutputPuml	createPlantUML	Performance verbessern
GUI_SWT	-	Entwicklerdokumentation (Installationsanleitung) für verwendetes Tool

Sonstiges:

- mehr Kommentare
- (Graphviz muss installiert sein, um PlantUML anzuzeigen)
- Javadocs schreiben!
- in gitconfig Name und Mail-Adresse anpassen! Wichtig für Benotung!
- Ordner für Unit-Tests ist srcTest
- im Ordner srcTest ein Unterordner “testfiles“ ertellen, in dem zusätzliche Testdateien landen

IV.10 Ergebnisse der Retrospektive

XXX

IV.11 Abschließende Einschätzung des Product-Owners

XXX

IV.12 Abschließende Einschätzung des Software-Architekten

XXX

IV.13 Abschließende Einschätzung des Team-Managers

XXX

V. Sprint 2

VI. Dokumentation

VI.1 Handbuch

Hier soll das Deutsche Benutzerhandbuch entstehen.

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