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

- Zielbestimmung

  PlantUML ist eine einfache, menschen-lesbare Speifikationssprache für die Erzeugung von UML-Diagrammen. Häufig werden allerdings auch UML-Diagramme zu bestehendem Quellcode benötigt, was durch eine gut unterstützte Generierung von PlantUML-Beschreibungen aus Java-Quelltext ermöglicht werden soll.

- Produkteinsatz

  Für den Einsatz in der Lehre ist die Generierung von Klassendiagrammen und Sequanzdiagrammen notwendig.

- Produktbeschreibung

  - eine jar-Datei oder eine Menge an java-Dateien ist einlesbar
  - der Quelltext ist zu analysieren und die identifizierten Klassen und Verknüpfungen sind anzuzeigen - einschließlich use-Beziehungen
  - der Nutzer kann entscheiden, welche Bestandteile in ein Klassen-Diagramm eingehen sollen
  - eine Voransicht des beschriebenen Diagramms ist zu integrieren
  - eine Unterstützung für das Layout ist ebenfalls anzubieten
  - ferner soll für den Aufruf einer oder mehrerer Methoden ein Sequanzdiagramm abgeleitet werden - auch hier soll der Nutzer die Möglichkeit haben, einzelne tiefere Aufrufe zu blockieren bzw. sich bei alternativen Pfaden für einen Pfad zu entscheiden

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?

  Die Zielgruppe des Projekts beinhaltet alle auf Java programmierenden Personen welche zudem mit UML-Diagrammen meist vorwiegend komplexere Softwareprojekte visualisieren wollen. Spezifischer ist das Programm jedoch an fortgeschrittene bis professionelle Programmierer gerichtet um Softwareprojekte möglichst einfach und nach Wünschen des Nutzers in PlantUML zu überführen.

- Worauf legt unsere Zielgruppe besonderen Wert?

  Die Zielgruppe fordert einerseits eine einfache Schnittstelle zwischen Quellcode und PlantUML um leicht und bedienerfreundlich automatisch UML-Diagramme zu erzeugen und gegebenenfalls Veränderung an der automatisch erzeugten Visualisierungen vorzunehmen.

- Was für einen Mehrwert bietet unser Produkt der Zielgruppe?

  Der Mehrwert des Projekts liegt bei der Füllung der Lücke zwischen PlantUML und Java-Quelltext, sodass einzelne oder eine Menge von Jar-Dateien automatisch eingelesen werden und in Voransicht dargestellt so bearbeitet werden können, dass der Nutzer die Entscheidungskraft darüber hat, welche Bestandteile in ein Klassen-Diagramm mit eingehen sollen. Zudem hat der Nutzer die Möglichkeit Sequenzdiagramme aus einer oder mehrerer Methoden zu erzeugen. Auch hier wird dem Nutzer eine Einscheidungsgewalt gewährt, bei dieser er bestimmte Aufrufe blockieren und sich bei alternativen Pfaden für einen der selbigen Entscheiden kann. Somit wird eine Diagramm mit Klassen, Verknüpfungen und und use-Beziehungen nach Vorstellungen des Nutzers erstellt.

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

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

- web server unter: http://www.plantuml.com/plantuml/ für kleine Test, zum ausprobieren

- Syntax
  Beginn mit: „@startuml“ Ende: „@enduml“
  Kein „;“ zur Abgrenzung
  Neue Zeile für neue Anweisung
  Bsp.:

```
@startuml
Bob -> Alice
@enduml
```

- Klassen
  - Werden durch Schlüsselwort „class“ eingeleitet
  - Kann Attribute und/oder Methoden erhalten
  - Bei längeren Klassen „...lange Klasse...“ as long
  - Sichtbarkeit der Klasse über Symbole Minus, Tilde, Plus und Doppelkreuz
  - Attribute und Merkmale können „Abstract“ oder „Static“-Merkmale bekommen über geschweifte Klammern

  Bsp. 1 - Einzelne Klasse:

```
@startuml
class ErsteKlasse
@enduml
```

  Bsp. 2 - Klasse mit langem Namen:

```
@startuml
class "wirklich alleralleraller ErsteKlasse" as long
@enduml
```

  Bsp. 3 - Klasse mit Methode und Datendeklaration:

```
@startuml
class ErsteKlasse : Name
@enduml
Bsp.:
@startuml
class ZweiteKlasse{
String name
Integer wert
void methode(String parameter1, Int parameter2)
}
@enduml
```

Bsp. 4 - Sichtbarkeit der Klassen:

```
Bsp.:
@startuml
class MeineKlasse{
- Private
# Protected
~ PackagePrivate
+ Public
}
@enduml
```

Bsp. 5 - Static / Abstract:

```
@startuml
class MeineKlasse{
     {static} String password
     {abstract} void methods()
     }
@enduml
```

Bsp. 6 - Abstrakte Klasse

```
@startuml
abstract class AbstrakteZweiteKlasse
@enduml
```

Bsp. 7 . Interface

```
@startuml
interface schoenesInterface
@enduml
```

- Klassen Verbinden
  - einfach
  - gerichtet in eine Richtung
  - in Beide Richtungen gerichtet

  Bsp. 1 - Einfache Verbindung

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse -- ZweiteKlasse
```

Bsp. 2 - Gerichtete Verbindung

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse --> ZweiteKlasse
@enduml
```

Bsp. 3 - In beide Richtungen gerichtet

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse <--> ZweiteKlasse
@enduml
```

Bsp. 4 - Lose-Verbindung

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse o-- ZweiteKlasse
@enduml
```

Bsp. 5 - Gebundene Klassen

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse *-- ZweiteKlasse
@enduml
```

Bsp. 6 - Vererbung Unterlass/Oberklasse

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse o--|> ZweiteKlasse
@enduml
```

Bsp. 7 - Notiz

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse -- ZweiteKlasse
note ‘‘Dies ist‘‘ as notiz1
ErsteKlasse .. notiz1
```

- Pfeile - kurze Pfeile/ lange Pfeile (-/—) unterschied Anordnung
  - für includes und extends (..)
  Bsp. 1 - kurz

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse -> ZweiteKlasse
@enduml
```

Bsp. 2 - lang

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse --> ZweiteKlasse
@enduml
```

Bsp. 3 - gestrichelt

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse..> ZweiteKlasse
@enduml
```

Bsp. 4 - include und extend Assoziation

```
Bsp2. gestrichelt - include und extend Assoziation:
@startuml
(A) .> (B): <<include>>
(B) .> (C): <<extend>>
@enduml
```

Bsp. 5 - Weitere Kombinationen zum Test auf www.plantuml.com/plantuml/

```
@startuml
Bob -> Alice: synchrone Nachricht von Bob an Alice
Bob ->> Alice: asynchrone Nachricht von Bob an Alice
Bob --> Alice: gestrichelte Linie als Antwortnachricht
Bob -\ Alice: Pfeilspitze ist nur oberhalb der Linie gezeichnet
Bob ->x Alice: verlorene Nachricht von Bob an Alice, x an Pfeilspitze
Bob \\- Alice: Pfeilspitze ist nur unterhalb angezeigt, jedoch offen
Bob //-- Alice: gestrichelte Linie mit offener Pfeilspitze oberhalb
Bob ->o Alice: Kreis am Ende der Pfeilspitze auf der Lebenslinie von Alice Bob <->
    Alice: bidirektionaler Pfeil
Bob <->o Alice: bidirektionaler Pfeil mit Kreis am Ende der Pfeilspitze
@enduml
```

- Beschriftungen der Verbindungen
  - einfache Annotationen durch „:“
  - mit Richtungsangabe

  Bsp. 1 - Simple Beschriftung

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse - ZweiteKlasse : Beschriftung mit Leerzeichen
@enduml
```

Bsp. 2 - Beschriftung mit Leserichtung

```
@startuml
class ErsteKlasse
class ZweiteKlasse
ErsteKlasse - ZweiteKlasse : Beschriftung mit Leserichtung <
@enduml
```

- Limitierung der Darstellung in PlantUML
  -keine direkten Verzweigungen, immer an Element gekoppelt
  - es können inkonsistente Zeichnungen entstehen
  Bsp. 1 - Vererbung im Kreis:

```
@startuml
class a
class b
a -|> b
b -|> a
note "Fehler" as notiz1 a .. notiz1
@enduml
```

- UML Erweiterungen von Interesse
  Unterstützung für:
  - Java API (plantuml.jar)
  - png from String -png from File -svg from String
  - Command Line/Terminal
  - Eclipse (Plug-In)
  - LaTex
  - Atom, GEdit, VIM, weitere txt-editoren...

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:

- Einlesen von Projekten aus verschiedenen Programmiersprachen
  - Beispielsweise:
    * Python
    * C++
    * C#
    * etc.
  - Nützlichkeit:

        * Erhöht Einsetzbarkeit des Tools enorm
        * Erhöht die Anzahl der möglichen Nutzer des Tools
    - Umsetzbarkeit:
        * Durch die XML-Schnittstelle muss alleinig der Parser angepasst werden
        * Den Parser ist allerdings die aufwendigste Klasse des Programms

- Erstellung eines Eclipse Plugins
    - Nützlichkeit:
        * Erhöht Usebility enorm
        * Erleichtert die Benutzung
        * Vergrößert die wahrscheinliche Anzahl der Nutzer durch einfache Einbindung
    - Umsetzbarkeit:
        * Umsetzung durch Umsetzung in Konkurenzprodukten erleichtert
        * Wahrscheinlich in geplantem Zeitraum umsetzbar

- Anpassen des Quellcodes durch Änderungen im erstellten UML Diagramm
    - Nützlichkeit:
        * Erhöht die Nutzbarkeit des Tools als ein Gesamtprodukt durch komplette Funktionalität
        * Gibt dem Tool eine flexiblere Nutzbarkeit
    - Umsetzbarkeit:
        * In vorgegebener Zeit schwer realisierbar
        * Mögliche Erweiterung als weiterführendes Projekt

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?

Workflow:

1. Wähle im Programm „Öffnen“ aus und suche im Explorer die Datei
2. Datei wird eingelesen und verarbeitet
3. Ich sehe das ganze UML Diagramm auf der einen Seite und den Quelltext auf der anderen
4. Im Quelltext kann ich einzelne Parts auswählen, wodurch kleinere UML Diagramme erstellt werden
5. Ich kann die Anordnung der dargestellten Elemente verändern
6. Nun besteht die Möglichkeit das Diagramm z.B. als png-Datei zu exportieren

GUI:

- Ganz oben wären Button zum minimieren, maximieren und schließen. Darunter eine Menüleiste mit verschiedenen Optionen um Quelltext und UML-Diagramme zu öffnen und speichern.
- Am linken Rand ist die Klassenstruktur, wie man sie z.B. in Eclipse hat dargestellt zur Koordination. Direkt daneben befindet sich ein Feld mit dem Quelltext, damit man sich direkt dort vergewissern kann, wie Klassen, die im UML-Diagramm dargestellt sind miteinander im Quelltext interagieren.
- Auf der linken Seite ist das UML Diagramm dargestellt.
- Die drei wichtigsten Knöpfe der GUI wären: Öffnen, Speichern und Diagramm bearbeiten

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.
Fehlende Kompetenzen für die Entwicklung des Produktes:

- Keine UML-Kenntnisse
- Keine Programmierkenntnisse welche über die bisherigen Anforderungen des Studiums hinausgehen
- Keine Erfahrungen mit größeren Gruppenarbeiten
- Keine bisherige GUI-Programmierung mit Hilfe von Tools
- Noch nie mit GIT gearbeitet
- Keine Kenntnisse der Fähigkeiten der Gruppenmitglieder

Genannte Stichpunkte treffen zwar meist nicht auf alle, aber dennoch auf einen Großteil der Gruppe zu. Im Laufe des Projekts werden die meisten fehlenden Kompetenzen automatisch wegfallen, da diese auf Erfahrungen aufbauen. Das Arbeiten mit GIT zum Beispiel wird sicherlich nach mehreren Benutzen einfacher.

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

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

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.
10.01.19	Müssen auch vorkompilierte .jar-Dateien eingelesen werden können?	Eine Dekompilierung von bereits vorkompilierten .jar-Dateien ist nicht erforderlich, in diesem Fall reicht es, wenn eine Fehlermeldung ausgegeben wird, dass bereits kompilierte Dateien ausgewählt wurden.
	Wie schlimm ist es, dass das Tool für die grafische Oberfläche nicht unter Linux läuft?	Am wichtigsten ist später zwar die Plattformunabhängigkeit des Produkts, dennoch sollte nach Möglichkeit sichergestellt werden, dass jedes der Entwicklungswerkzeuge jedem Entwickler zur Verfügung steht, egal, auf welchem Betriebssystem entwickelt wird.

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 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.

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

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

Swing ist eines der meistgenutzten GUI-Frameworks für Java, war bis 2014 ein Standard-Tool zur GUI Entwicklung und hat aufgrund dessen eine große Community hinter sich und man findet viel Hilfestellungen für Swing im Internet.

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

Aufgrund der erst kurzen Zeit, in der JavaFX zur Verfügung steht, gibt es hier viel weniger Hilfe online im Vergleich zu Swing. JavaFX gilt allerdings als der neue Standard in der Java GUI Entwicklung und es gibt sehr viele Developer, die von Swing zu JavaFX umsteigen.

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.1.6 Softwareverbreitung unter Windows

- Beschreibung

  Um eine unbeaufsichtigte Installation von Software zu gewährleisten benötigt unser Projekt für Windows ein installierbares Softwarepaket. Um ein solches Softwarepaket zu erstellen, benötigt man die Hilfe eines Installers.
- Installer-Typen
  - Kriterien

    Kriterien bezüglich der Installer sind zum einen, ob selbige mit Open-Source arbeiten. Zudem sollten die Installer im Bezug auf das nicht-kommerzielle Projekt nicht kostenpflichtig sein. Im Generellen sollte der ideale Installer außerdem eine einfache Handhabung innehaben und im Spezifischen die Überprüfung, ob die JRE installiert ist, und gegebenenfalls die Installation derselbigen anbieten.
  - Nullsoft Scriptable Install System (NSIS)

    NSIS bietet ein kostenlosen aber sehr flexiblen wie auch minimalen Installer. Jedoch sind durch Open-Source bereits viele Plugins vorhanden. Auch hier ist eine Überprüfung und Installation der JRE möglich. Der Compiler der NSIS's ist jedoch recht primitiv gestaltet und ein intuitives Verständnis des Codes gestaltet sich durch fehlendes Syntax-Highlighting schwer.

  - Inno Setup

    Das **Inno Setup** ist ein Open-Source-unterstützender Installer welcher kostenlos downloadbar ist. Der Installer kommt mit einem übersichtlichen Compiler, einfacher und gut strukturierter Syntax, wie einer Code-Sektion in welcher komplexe Vorgänge mit Pascal programmiert werden können, und zudem mit vorgefertigten Beispielen zu bestimmten Software-Typen. Auch eine Überprüfung und eventuelle Installation der JRE ist umsetzbar.

- Fazit

  Nach Betrachtung beider Installer genügen beide Typen vollkommen den gestellten Anforderungen. Jedoch ist das *Inno Setup* durch übersichtlichen Compiler und intuitiver Handhabung meine persönliche Empfehlung im Sinne des Softwareprojekts.

II.1.7 Exceptions

Mit Exceptions reagiert man auf Fehler und unerwartete Situationen während der Ausführung eines Programms. Diese werden mit einem try-Block erstellt und mit einem catch-Block abgefangen. Es existieren viele Möglichkeiten wodurch Exception auftreten können. Möglich sind beispielsweise überschrittene Arraygrenzen, Zugriff auf nicht erzeugte Objekte oder fehlerhafte Typkonvertierungen. Mit dem Code throw new Exception(); lässt sich bewusst eine Exception erstellen.
In den Java-Bibliotheken gibt es bereits viele Exceptiontypen. Allerdings ist es auch möglich eigene als Unterklasse der Exceptionklasse zu erstellen, wenn die vorhandenen nicht ausreichen.

Benötigte Exceptions für unser Projekt:
Für die Konsolenanwendung wären Exceptions sinnvoll. Mit diesen könnte man beispielsweise fehlerhafte Pfadangaben abfangen. Da in der GUI die Dateien über den Explorer eingelesen werden, sollten dort keine falschen Pfade zustande kommen.
Beispiel für Exceptions:

```
public class Main
{
   public static void main(String[] args)
   {
      int[] numberArray = { 1, 2, 0 };
      try
      {
         //System.out.println(1 / numberArray[2]);
         System.out.println(numberArray[3]);
      }
      catch (ArithmeticException exception)
      {
         System.out.println("Nicht durch Null teilen!");
      }
      catch (Exception e)
      {
         System.out.println("Fehler: " + e);
      }
      System.out.println("Programm wird trotz Fehler weiterhin ausgefuehrt");
   }
}
```

Da das Programm auf eine Stelle im numberArry zugreifen möchte, welche nicht existiert, kommt es zu der Fehlermeldung. Kommentiert man die zweite anstelle der ersten Systemausgabe aus, so

kommt es zu einer anderen Fehlermeldung: „Nicht durch Null teilen!“ Das Beispiel zeigt, dass durch das Catchen verschiedener Fehlertypen verschiedene Befehle ausgeführt werden können.

II.2 Entscheidungen des Technologieworkshops

XXX

II.3 Überblick über Architektur

XXX

II.4 Definierte Schnittstellen

XXX

II.5 Liste der Architekturentscheidungen

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. 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, 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.
Sonstige Konventionen:

- Variablen und Instanzen beginnen kleingeschrieben

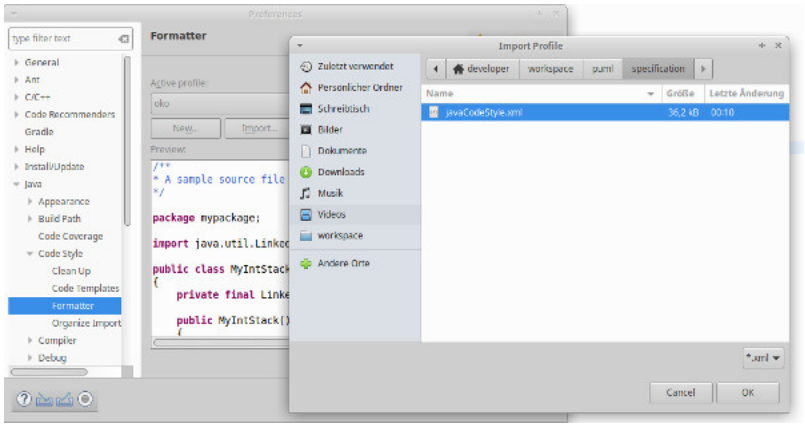

Abbildung 2: Code-Style in Eclipse importieren

- 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.

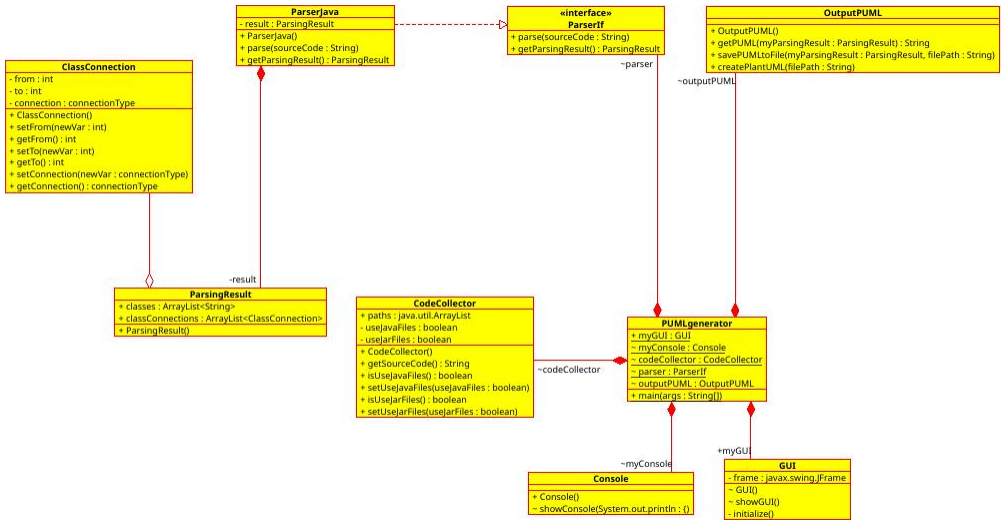

Abbildung 3: Klassendiagramm des Sprints

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

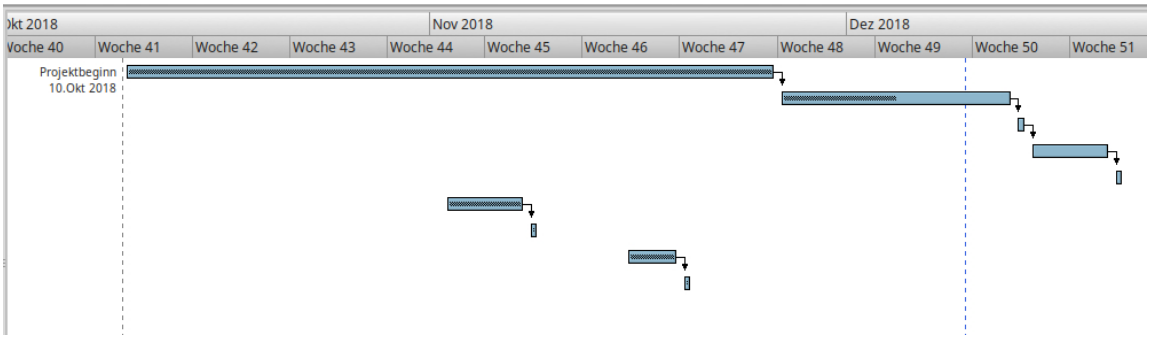

Abbildung 4: Gantt-Diagramm für Sprint 1

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 weiß, welcher Teil des Programms zu implementieren ist.

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

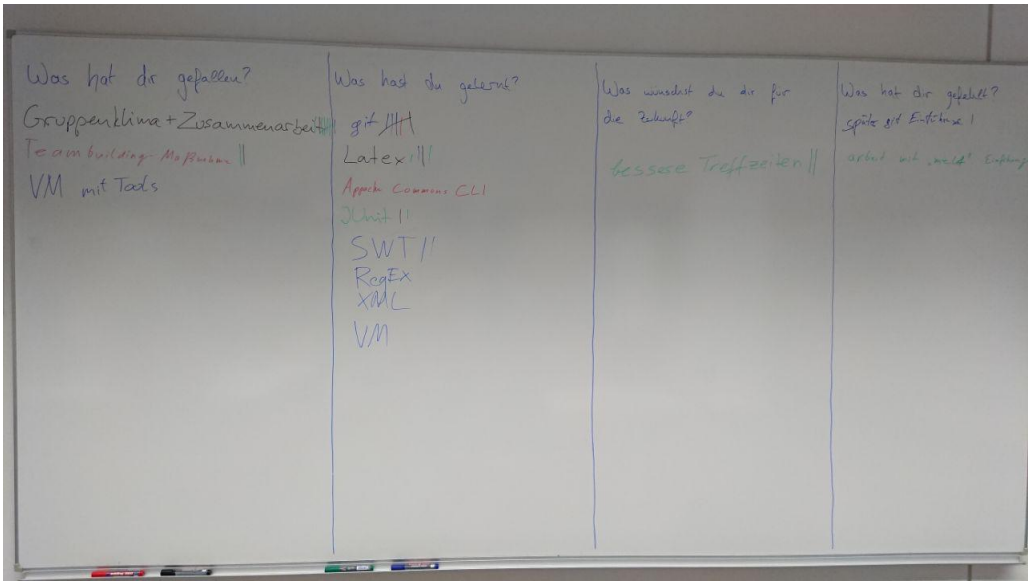

Abbildung 5: Anmerkungen der Teammitglieder zum bisherigen Verlauf des Projekts

Die Retrospektive schloss, sowohl was den Lernerfolg als auch die Kommunikation im Team angeht, mit einer positiven Bilanz. Gelobt wurde besonders das Gruppenklima und die Zusammenarbeit sowie die Verteilung der fertig eingerichteten Virtual Machine. Kollektive Lernerfolge sind besonders in den Bereichen Git und Latex zu verzeichnen, daneben decken die genannten spezialisierten Bereiche die Themenfelder ab, die den jeweiligen Gruppen zugeteilt wurden. Für die Zukunft hofft das Team auf günstigere Zeiten für die Projekttreffen. Bemängelt wurde, dass es bei der Einführung in das Arbeiten mit Git allgemein Defizite gab. Spezifisch machte besonders die Bedienung des Meld-Tools Schwierigkeiten. Es wurde deshalb beschlossen, auf diese Probleme in einem der folgenden Gruppentreffen noch einmal ausführlich einzugehen.
Befürchtet wurde vor allen Dingen, dass im nächsten Semester mehr Fehler auftreten werden, als das Team zunächst vermutet hätte. Um das Risiko zu verringern, dass sich diese Befürchtungen bewahrheiten, wurden bereits bekannte Bugs zusammengetragen und in Jira gestellt. Ziel des zweiten Sprints ist die vollständige Implementierung aller für den ersten Sprint definierten Use Cases (sofern noch nicht erfolgt) wie auch das Beheben aller bisher dokumentierten Bugs. Die Praxis des Pair Programmings wird zunächst beibehalten. Angestrebt ist dennoch eine bessere Dokumentierung des Quellcodes, auch, um eine potenzielle Umverteilung der Teammitglieder zu erleichtern. Im nächsten Semester soll für die Teammitglieder die Möglichkeit bestehen, auf Wunsch in einen anderen Teilbereich des Projektes „hineinzuschnuppern“. Auch die in Jira angelegten Issues sollen besser getrennt werden, um sie auch an Einzelpersonen zuweisen zu können.

IV.11 Abschließende Einschätzung des Product-Owners

Insgesamt konnten die meisten für den ersten Sprint definierten Use Cases implementiert werden. Damit existiert bereits eine minimale, lauffähige Version des Programms. Werden im nächsten Sprint die noch nicht vollständig im ersten Sprint implementierten Funktionen fertiggestellt sowie enthaltene Bugs entfernt, ist eine solide Grundlage für die weitere Entwicklung des Produkts gelegt.

IV.12 Abschließende Einschätzung des Software-Architekten

XXX

IV.13 Abschließende Einschätzung des Team-Managers

Für dieses Projektteam ist die Rolle des Team-Managers nicht vergeben. Von den Ergebnissen der Retrospektive ausgehend lässt sich allerdings annehmen, dass die bisherige Vorgehensweise bei der Organisation des Projekts sowie der Durchführung der Treffen und des ersten Sprints grundsätzlich ein guter Ansatz zu sein scheint.

V. Sprint 2

V.1 Ziel des Sprints

Dieser Sprint ist ausschließlich dazu gedacht, im Verlauf des ersten Sprints identifizierte und noch nicht behobene Bugs zu entfernen. Es wurden bewusst keine neuen User-Stories (für den Benutzer) im Sprint-Backlog definiert. Der Fokus liegt darauf, die im ersten Sprint geschaffene Basis noch einmal zu stabilisieren.

V.2 User-Stories des Sprint-Backlogs

V.2.1 Reduzierung von Bugs

Als Softwarearchitekt und Product Owner wünschen wir uns, dass möglichst wenige Bugs auftreten, um die spätere Weiterentwicklung und damit die uneingeschränkte Funktionalität des Produkts nicht zu gefährden.

V.3 Zeitliche Planung

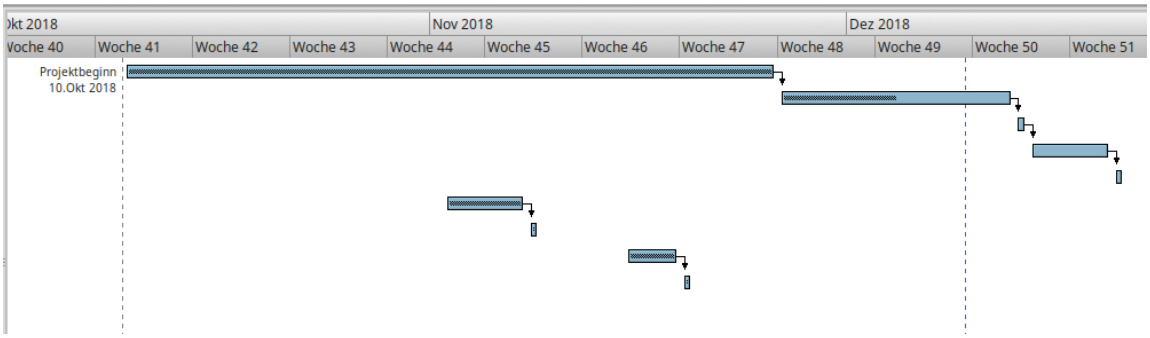

Abbildung 6: Gantt-Diagramm für Sprint 1

V.4 Liste der durchgeführten Meetings

- Planning-Meeting (00.00.2019)
- Zwischen-Meeting (00.00.2019)
- Review-Meeting (00.00.2019)

V.5 Ergebnisse des Planning-Meetings

Der zweite Sprint wird zeitlich in der letzten Woche der Semesterferien begonnen und bis zum Ende der ersten Woche der Vorlesungszeit gehen. Dies wurde mit den Teammitgliedern besprochen. Hauptziel des Sprints ist ein sauberer Stand, mit dem ab dem kommenden Sommersemester weitergearbeitet werden kann.

V.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					

V.7 Konkrete Code-Qualität im Sprint

Die Code-Qualität wurde durch den Fokus auf die Behebung bestehender Bugs verbessert.

V.8 Konkrete Test-Überdeckung im Sprint

Die Test-Überdeckung in diesem Sprint war überdurchschnittlich.

V.9 Ergebnisse des Reviews

Klasse	Methode	Anmerkungen

V.10 Ergebnisse der Retrospektive

Die Retrospektive schloss mit einer positiven Bilanz. (...)

V.11 Abschließende Einschätzung des Product-Owners

In diesem Sprint wurden einige kritische Bugs behoben und somit User Stories des letzten Sprint-Backlogs noch vervollständigt. Zu hoffen bleibt trotzdem, dass in allen folgenden Sprints auch neue Funktionalität hinzugefügt wird.

V.12 Abschließende Einschätzung des Software-Architekten

XXX

V.13 Abschließende Einschätzung des Team-Managers

Insgesamt kann positiv herausgehoben werden, dass sich die Teammitglieder geschlossen dazu bereit erklärten, Teil ihrer Semesterferien für den zweiten Sprint zu opfern. Die Motivation, das Produkt weiter voranzubringen, scheint derzeit ungebrochen.

VI. Sprint 3

VI.1 Ziel des Sprints

Die bestehenden Funktionen des Programms sollen um die Möglichkeit ergänzt werden, neben Klassendiagrammen auch Sequenzdiagramme erstellen zu können. Dafür muss die Struktur der verarbeiteten Daten, also auch der Code zum Parsen, angepasst werden. Geplant ist eine Repräsentation des eingelesenen Codes als zentrale XML-Datei, die je nach Anwendungszweck wiederum die Basis für zwei unterschiedliche XML-Dateien ist.

VI.2 User-Stories des Sprint-Backlogs

VI.2.1 Sequenzdiagramme

Auswahl des zu erstellenden Diagramms Als Benutzer wünsche ich mir, dass eine Auswahl zwischen Klassen- und Sequenzdiagrammen möglich ist, damit ich diese je nach meinen Bedürfnissen generieren kann.

Generierung von Sequenzdiagrammen Als Benutzer wünsche ich mir, Sequenzdiagramme erstellen zu können, um einen Überblick über die Abläufe meines Programms zu erhalten.

VI.2.2 Interne Struktur

Parserfunktionalität Als Softwarearchitekt wünsche ich mir, dass der Parser effizient funktioniert, um die benötigten Daten ohne Schwierigkeiten auszulesen.

Dateiformat XML Als Softwarearchitekt wünsche ich mir, dass der übergebene Code zur weiteren Verarbeitung in verschiedene XML-Dateien umgewandelt wird.

VI.3 Zeitliche Planung

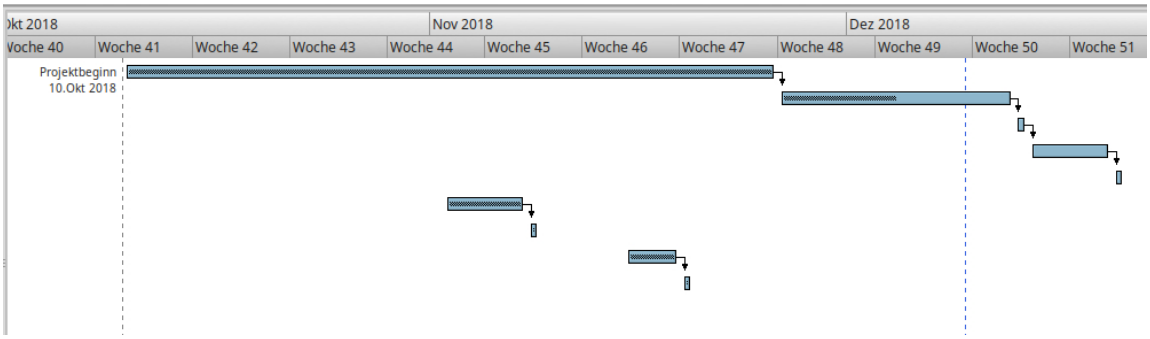

Abbildung 7: Gantt-Diagramm für Sprint 1

VI.4 Liste der durchgeführten Meetings

- Planning-Meeting (15.04.2019)
- Parser-Besprechung (18.04.2019)
- Parser-Besprechung (24.04.2019)
- Zwischen-Meeting (25.04.2019)
- Review-Meeting (29.4.2019)

VI.5 Ergebnisse des Planning-Meetings

Dem gesamten Team ist die geplante Grundstruktur des Programms bekannt. Jeder weiß, welcher Teil des Programms zu implementieren ist.

VI.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					

VI.7 Konkrete Code-Qualität im Sprint

Die Code-Qualität ist bisher gleichmäßig. Da die Umstrukturierung des Parsers besondere Sorgfalt erfordert, wurden zusätzliche Treffen mit den verantwortlichen Teammitgliedern angesetzt.

VI.8 Konkrete Test-Überdeckung im Sprint

Die Test-Überdeckung in diesem Sprint fiel eher niedrig aus, da für die neuen Funktionen erst passende Testdateien erstellt bzw. bestehende Testdateien angepasst werden müssen.

VI.9 Ergebnisse des Reviews

Klasse	Methode	Anmerkungen

Sonstiges:

- Zum Testen des überarbeiteten Parsers müssen die Testdateien angepasst werden

VI.10 Ergebnisse der Retrospektive

Die Retrospektive schloss mit einer gemischten Bilanz. Besonders die Umstrukturierung des Parsers warf viele Fragen auf, da nicht nur der Code des Parsers selbst, sondern auch alle Schnittstellen angepasst werden mussten. Hier zeichnete sich eine leichte Unzufriedenheit ab, da Teile der bisher geleisteten Arbeit nun umgeschrieben werden. Für den nächsten Sprint ist es daher wünschenswert, die Erzeugung der neuen XML-Dateien sowie darauf aufbauend der Sequenzdiagramme voranzutreiben, damit hier möglichst bald ein Erfolg zu verzeichnen ist.

VI.11 Abschließende Einschätzung des Product-Owners

Insgesamt ist ein beachtlicher Teil der definierten Sprintziele noch in Bearbeitung. Da es sich um einen kurzen Sprint handelte und zudem große Teile des Codes geändert werden müssen, um die Generation von Sequenzdiagrammen vorzubereiten, ist dies nachvollziehbar.

VI.12 Abschließende Einschätzung des Software-Architekten

XXX

VI.13 Abschließende Einschätzung des Team-Managers

Vom allgemeinen Sprintverlauf ausgehend wird das Erstellen von Sequenzdiagrammen die größte Herausforderung des Projekts werden. Damit die betreffenden Teammitglieder im Vergleich nicht zu viel Zeit in den Parser investieren müssen, wurden zwei zusätzliche Treffen mit dem Softwarearchitekten angesetzt, bei denen die notwendigen Voraussetzungen besprochen und die Methodik erörtert wurde.

VII. Sprint 4

VIII. Dokumentation

VIII.1 Handbuch

VIII.1.1 Kommandozeilenaufruf

Aufruf:
PUMLgenerator [-c -i inputPathes -o outputPathes (-cc | -s | -int | -cs entryClass entryMethod) [-ijar, -ijava]]

- Wird das Programm ohne Parameter aufgerufen öffnet sich die grafische Oberfläche
- -c wird verwendet um einen Konsolen-Aufruf zu initiieren
- Nach -c müssen mit -i die Eingabepfade, mit -o der Ausgabepfad und je nach gewünschter Aktion -cc, s, -int oder -cs stehen
- mit -ijar und -ijava können bestimmte Dateitypen ignoriert werden
- interaktiver Modus
  - Fragt ab welcher Typ von Diagramm erzeugt werden soll
  - Listet bei einem Sequenzdiagramm alle Klassen und Methoden auf und ermöglicht die Auswahl des Einstiegspunktes

Parameter	Übergabewerte	Info	Be
-c		Konsolenaufruf	keine gra
-i	inputPathes	Input	durch Strichpunkt getrennte Liste der P
-o	outputPath	Output	muss ein Pfad zu einem Ordner sein. Hier soll
-ijar		Ignore Jar	.jar-Datei
-ijava		Ignore Java	.java-Datei
-cc		create Classdiagram	erzeugt ein
-cs	entryClass, entryMethod	create Sequencediagram	erzeugt ein
-s		show	listet alle Klas
-int		interactive	intera

VIII.2 Installationsanleitung

XXX

VIII.3 Software-Lizenz

XXX

IX. Projektabschluss

IX.1 Protokoll der Abnahme und Inbetriebnahme beim Kunden

XXX

IX.2 Präsentation auf der Messe

Poster, Bericht

IX.3 Abschließende Einschätzung durch Product-Owner

XXX

IX.4 Abschließende Einschätzung durch Software-Architekt

XXX

IX.5 Abschließende Einschätzung durch Team-Manager

XXX