

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 | 4 |
| I.1 | Initiale Kundenvorgaben | 4 |
| I.2 | Produktvision | 5 |
| I.3 | Liste der funktionalen Anforderungen | 14 |
| I.4 | Liste der nicht-funktionalen Anforderungen | 15 |
| I.5 | Weitere Zuarbeiten zum Produktvisions-Workshop | 16 |
| I.6 | Zuarbeit von Autor X | 16 |
| I.7 | Zuarbeit von Autor Y | 16 |
| I.8 | Risikoanalyse | 16 |
| I.9 | Liste der Kundengespräche mit Ergebnissen | 17 |
| II | Architektur und Entwurf | 17 |
| II.1 | Zuarbeiten der Teammitglieder | 17 |
| II.2 | Entscheidungen des Technologieworkshops | 28 |
| II.3 | Überblick über Architektur | 28 |
| II.4 | Definierte Schnittstellen | 28 |
| II.5 | Liste der Architekturentscheidungen | 29 |
| III | Prozess- und Implementationsvorgaben | 29 |
| III.1 | Definition of Done | 29 |
| III.2 | Coding Style | 29 |
| III.3 | Zu nutzende Werkzeuge | 30 |
| IV | Sprint 1 | 30 |
| IV.1 | Ziel des Sprints | 30 |
| IV.2 | User-Stories des Sprint-Backlogs | 30 |
| IV.3 | Zeitliche Planung | 31 |
| IV.4 | Liste der durchgeführten Meetings | 31 |
| IV.5 | Ergebnisse des Planning-Meetings | 31 |
| IV.6 | Aufgewendete Arbeitszeit pro Person+Arbeitspaket | 32 |
| IV.7 | Konkrete Code-Qualität im Sprint | 32 |
| IV.8 | Konkrete Test-Überdeckung im Sprint | 32 |
| IV.9 | Ergebnisse des Reviews | 32 |
| IV.10 | Ergebnisse der Retrospektive | 32 |
| IV.11 | Abschließende Einschätzung des Product-Owners | 33 |
| IV.12 | Abschließende Einschätzung des Software-Architekten | 33 |
| IV.13 | Abschließende Einschätzung des Team-Managers | 34 |
| V | Sprint 2 | 34 |
| V.1 | Ziel des Sprints | 34 |
| V.2 | User-Stories des Sprint-Backlogs | 34 |
| V.3 | Zeitliche Planung | 34 |
| V.4 | Liste der durchgeführten Meetings | 34 |

| | | |
|---------|---|----|
| V.5 | Ergebnisse des Planning-Meetings | 35 |
| V.6 | Aufgewendete Arbeitszeit pro Person+Arbeitspaket | 35 |
| V.7 | Konkrete Code-Qualität im Sprint | 35 |
| V.8 | Konkrete Test-Überdeckung im Sprint | 35 |
| V.9 | Ergebnisse des Reviews | 35 |
| V.10 | Ergebnisse der Retrospektive | 36 |
| V.11 | Abschließende Einschätzung des Product-Owners | 36 |
| V.12 | Abschließende Einschätzung des Software-Architekten | 36 |
| V.13 | Abschließende Einschätzung des Team-Managers | 36 |
| VI | Sprint 3 | 36 |
| VI.1 | Ziel des Sprints | 36 |
| VI.2 | User-Stories des Sprint-Backlogs | 36 |
| VI.3 | Kontroll-Diagramm | 38 |
| VI.4 | Liste der durchgeführten Meetings | 38 |
| VI.5 | Ergebnisse des Planning-Meetings | 38 |
| VI.6 | Aufgewendete Arbeitszeit pro Person+Arbeitspaket | 38 |
| VI.7 | Konkrete Code-Qualität im Sprint | 38 |
| VI.8 | Konkrete Test-Überdeckung im Sprint | 38 |
| VI.9 | Ergebnisse des Reviews | 39 |
| VI.10 | Ergebnisse der Retrospektive | 39 |
| VI.11 | Abschließende Einschätzung des Product-Owners | 39 |
| VI.12 | Abschließende Einschätzung des Software-Architekten | 39 |
| VI.13 | Abschließende Einschätzung des Team-Managers | 39 |
| VII | Sprint 4 | 39 |
| VII.1 | Ziel des Sprints | 39 |
| VII.2 | User-Stories des Sprint-Backlogs | 39 |
| VII.3 | Zeitliche Planung | 40 |
| VII.4 | Liste der durchgeführten Meetings | 40 |
| VII.5 | Ergebnisse des Planning-Meetings | 40 |
| VII.6 | Aufgewendete Arbeitszeit pro Person+Arbeitspaket | 40 |
| VII.7 | Konkrete Code-Qualität im Sprint | 41 |
| VII.8 | Konkrete Test-Überdeckung im Sprint | 41 |
| VII.9 | Ergebnisse des Reviews | 41 |
| VII.10 | Abschließende Einschätzung des Product-Owners | 42 |
| VII.11 | Abschließende Einschätzung des Software-Architekten | 42 |
| VII.12 | Abschließende Einschätzung des Team-Managers | 42 |
| VIII | Sprint 6 | 42 |
| VIII.1 | Ziel des Sprints | 42 |
| VIII.2 | User-Stories des Sprint-Backlogs | 42 |
| VIII.3 | Zeitliche Planung | 42 |
| VIII.4 | Liste der durchgeführten Meetings | 42 |
| VIII.5 | Ergebnisse des Planning-Meetings | 42 |
| VIII.6 | Aufgewendete Arbeitszeit pro Person+Arbeitspaket | 43 |
| VIII.7 | Konkrete Code-Qualität im Sprint | 43 |
| VIII.8 | Konkrete Test-Überdeckung im Sprint | 43 |
| VIII.9 | Ergebnisse des Reviews | 43 |
| VIII.10 | Ergebnisse der Retrospektive | 44 |
| VIII.11 | Abschließende Einschätzung des Product-Owners | 44 |
| VIII.12 | Abschließende Einschätzung des Software-Architekten | 44 |
| VIII.13 | Abschließende Einschätzung des Team-Managers | 44 |

| | | |
|-------|--|----|
| IX | Sprint 7 | 44 |
| IX.1 | Ziel des Sprints | 44 |
| IX.2 | User-Stories des Sprint-Backlogs | 44 |
| IX.3 | Zeitliche Planung | 45 |
| IX.4 | Liste der durchgeführten Meetings | 45 |
| IX.5 | Ergebnisse des Planning-Meetings | 45 |
| IX.6 | Aufgewendete Arbeitszeit pro Person+Arbeitspaket | 46 |
| IX.7 | Konkrete Code-Qualität im Sprint | 46 |
| IX.8 | Konkrete Test-Überdeckung im Sprint | 46 |
| IX.9 | Ergebnisse des Reviews | 47 |
| IX.10 | Abschließende Einschätzung des Product-Owners | 47 |
| IX.11 | Abschließende Einschätzung des Software-Architekten | 47 |
| IX.12 | Abschließende Einschätzung des Team-Managers | 47 |
| X | Dokumentation | 48 |
| X.1 | Handbuch | 48 |
| X.2 | Software-Lizenz | 49 |
| XI | Projektabschluss | 50 |
| XI.1 | Protokoll der Abnahme und Inbetriebnahme beim Kunden | 50 |
| XI.2 | Präsentation auf der Messe | 50 |
| XI.3 | Abschließende Einschätzung durch Product-Owner | 50 |
| XI.4 | Abschließende Einschätzung durch Software-Architekt | 50 |
| XI.5 | Abschließende Einschätzung durch Team-Manager | 51 |

I. ANFORDERUNGSSPEZIFIKATION

I.1 Initiale Kundenvorgaben

- Zielbestimmung

PlantUML ist eine einfache, menschen-lesbare Spezifikationssprache 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 Sequenzdiagrammen 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 Sequenzdiagramm 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 Klassendiagramms 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
- Unterstützung für das Layout:
 - Layout muss automatisch konfigurierbar sein
 - Layout muss die Möglichkeit haben manuell konfiguriert werden zu können
- Beide Diagrammartentypen sollen als String oder als Textdatei ausgegeben werden können

Aufgaben für den 1. Sprint:

- Erstellen eines ersten GUIs
- 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 Entscheidungsgewalt 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 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.

1.2.3 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 Practices" identifiziert 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 Useability enorm
 - * Erleichtert die Benutzung
 - * Vergrößert die wahrscheinliche Anzahl der Nutzer durch einfache Einbindung
 - Umsetzbarkeit:
 - * Umsetzung durch Umsetzung in Konkurrenzprodukten 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
 - java.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 "Baustellen" zeigen 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 $\geq 50\%$
- 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, P U M L 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

| Wahrscheinlichkeit | Auswirkung | Gesamt | Risiko | Maßnahmen |
|--------------------|------------|--------|---------------------------------------|---|
| 5 | 5 | 25 | Entwickler fällt aus | <ul style="list-style-type: none"> 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 | <ul style="list-style-type: none"> 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 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

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

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);

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

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 JUnit

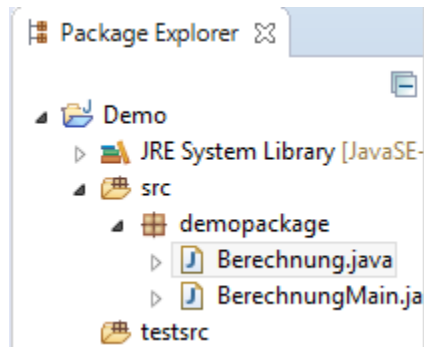
JUnit ist ein Framework zum Testen von Java-Programmen, das besonders für automatisierte Unit-Tests einzelner Units (Klassen oder Methoden) geeignet ist. Die aktuelle Version ist JUnit 5, bestehend aus JUnit Platform, JUnit Jupiter und JUnit Vintage.

In JUnit Jupiter werden die Tests geschrieben, in JUnit Platform ausgeführt und mit JUnit Vintage können ältere Versionen von JUnit - JUnit 3 und JUnit4 - durchgeführt werden.

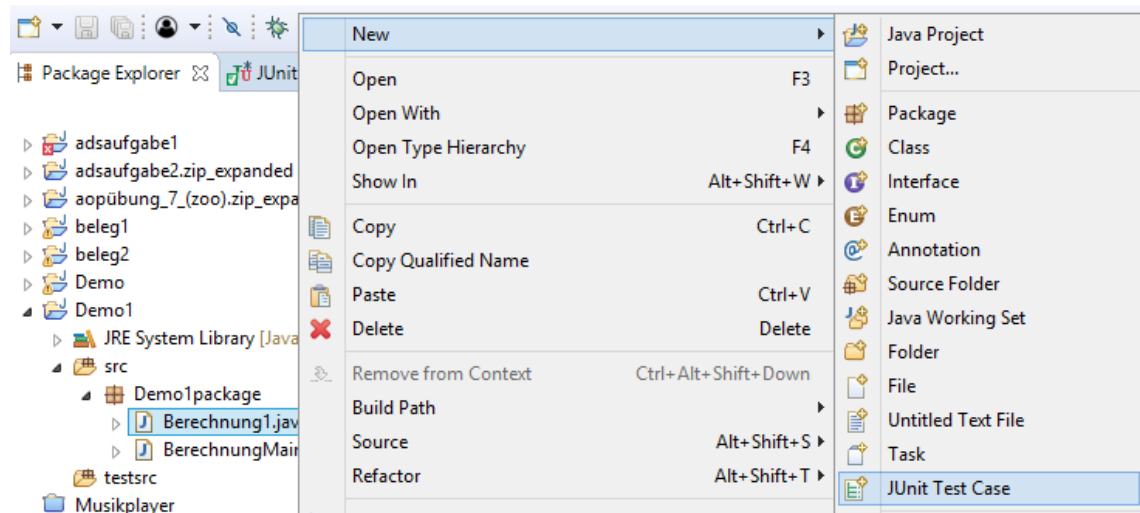
JUnit muss für die Nutzung in Eclipse nicht extra installiert werden, aber die JUnit 5 Library muss in den Build Path des jeweiligen Projektes aufgenommen werden. Von Eclipse gibt es auch eine Anleitung dazu ¹

Zuerst einmal wird das Projekt erstellt mit einem zusätzlichen Source-Ordner für den Test:

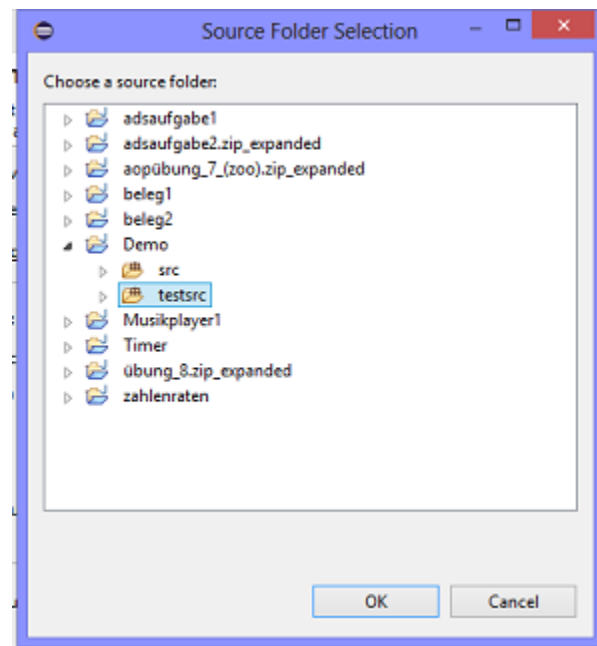
¹<https://www.eclipse.org/eclipse/news/4.7.1a/#junit-5-support>



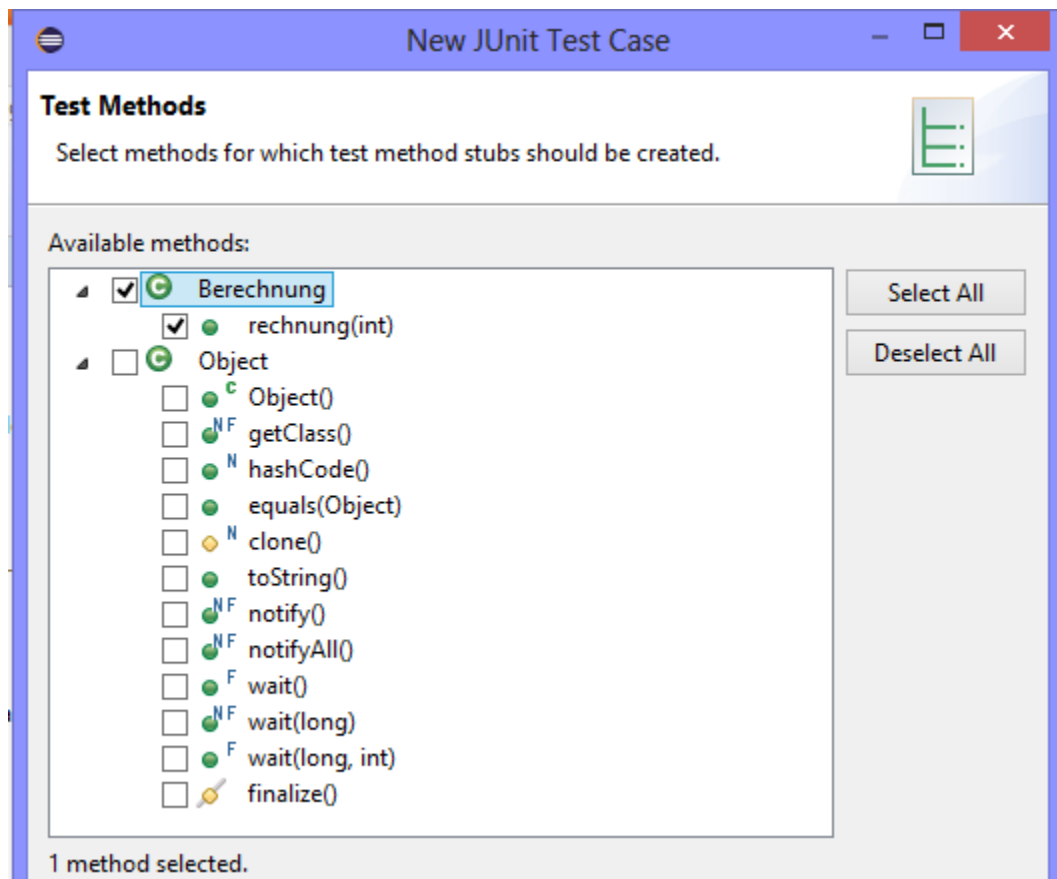
Mit Rechtsklick auf die Klasse, die getestet werden soll, kann ihr ein JUnit Test Case zugewiesen werden:



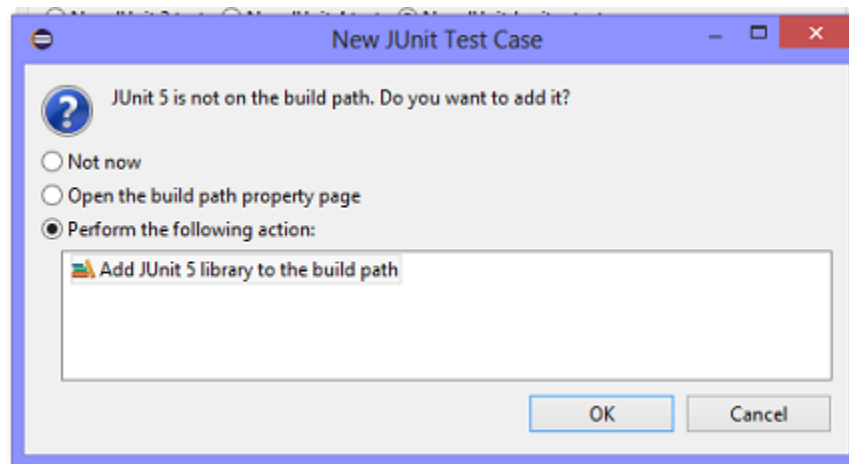
Als Source-Folder wird dabei der dafür angelegte gewählt:



Anschließend klickt man auf die Schaltfläche Next > und kann auswählen, was getestet werden soll:



Nach Klicken auf den Button Finish muss noch die JUnit 5 library zum Build Path hinzugefügt werden:



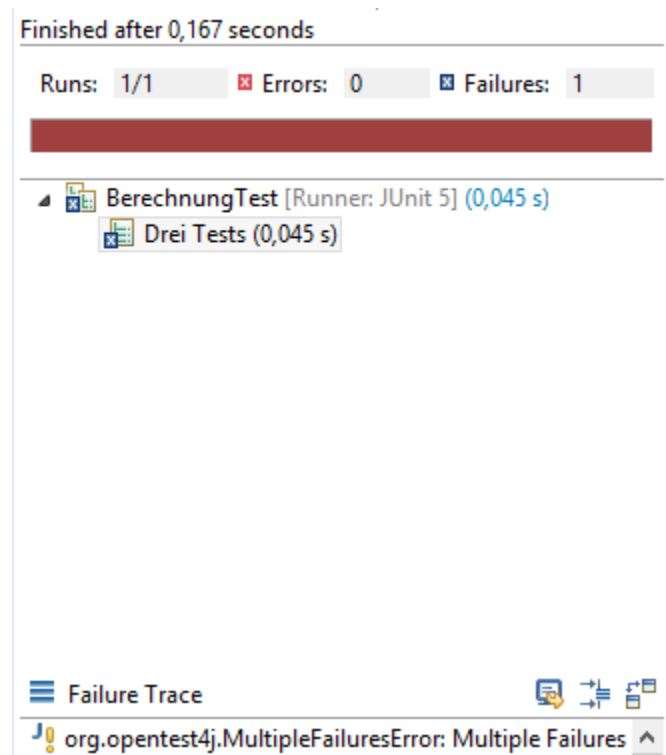
Der automatisch erstellte Test kann jetzt an das eigene Projekt angepasst werden. In einem Test werden die Parameter (hier int a), das erwartete und die Berechnung des tatsächlichen Ergebnisses angegeben. Mit `assertEquals(expected, actual);` werden die beiden Ergebnisse miteinander verglichen.

```
assertAll(
    () -> {
        // Test #1
        int a = 6;
        double expectederg = 40;
        double actualerg = classUnderTest.rechnung(a);
        assertEquals(expectederg, actualerg);
    },
    () -> {
        // Test #2
        int a = -3;
        double expectederg = -20;
        double actualerg = classUnderTest.rechnung(a);
        assertEquals(expectederg, actualerg);
    },
    ...
);
```

Mit dieser Anweisung kann festgelegt werden, dass eine Funktion vor jedem einzelnen Test durchgeführt werden. @BeforeEach public void SetUp() throws Exception classUnderTest = new Berechnung();

Weitere Anweisungen wie diese sind zum Beispiel @BeforeAll (= bevor irgendein Test durchgeführt wird), @AfterEach (nach jedem Test) und @AfterAll (nach allen Tests).

Nach Durchführung eines Testes erscheint links anstelle des Package Explorer ein JUnit Tab mit dem Ergebnis des Tests.



Bei einem nicht bestandenem Test ist der Balken rot und unten wird klein angezeigt, warum der Test nicht bestanden wurde.

Mit den drei Buttons rechts über dem Text kann man die Ansicht ändern (vlnr):
 Bei bestandenem Test ist der Balken grün.
 Ein etwas ausführlicheres Tutorial (auf Englisch) gibt es hier ²

II.1.4 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 p = Pattern.compile(regex);
Pattern.matches(regex, this);  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 verschiedenen 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] |

²https://www.youtube.com/watch?v=QNv_AQk6WWI

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.1.5 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
 - Etabliertes Austauschformat
 - Erweiterbar
- Nachteile gegenüber JSON:
 - Enthält viel "Ballast" der für Datenaustausch nicht nötig ist
 - Datenvolumen relativ hoch
 - Komplexe Syntax

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <Softwareprojekt jahr="2018">
4   <Gruppe gID="3">
5     <Master>
6       <Projektowner mID="1"> Anna
7     </Projektowner>
8       <Projektleader mID="2"> Oke
9     </Projektleader>
10    </Master>
11    <Bachelor>
12      <!-- Hier Mitglieder einfügen -->
13    </Bachelor>
14  </Gruppe>
15 </Softwareprojekt>

```

Abbildung 1: Beispielcode XML

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.1.6 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.1.7 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 derselben anbieten.
 - Nullsoft Scriptable Install System (NSIS)
NSIS bietet ein kostenloses aber sehr flexibles 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.1.8 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 ausgeführt");
    }
}
```

Da das Programm auf eine Stelle im numberArray 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 Technologiewerkshops

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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • ... |

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 Zuge-

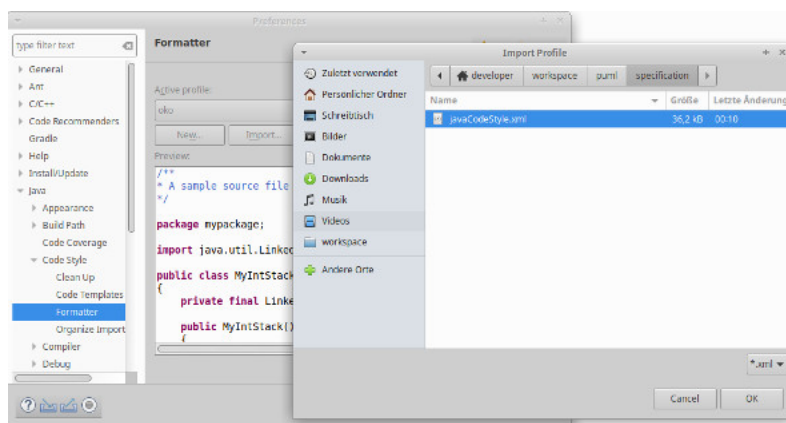


Abbildung 2: Code-Style in Eclipse importieren

hörigkeit der Schließenden Klammer mit einem Kommentar zu Kennzeichnen.
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.

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.

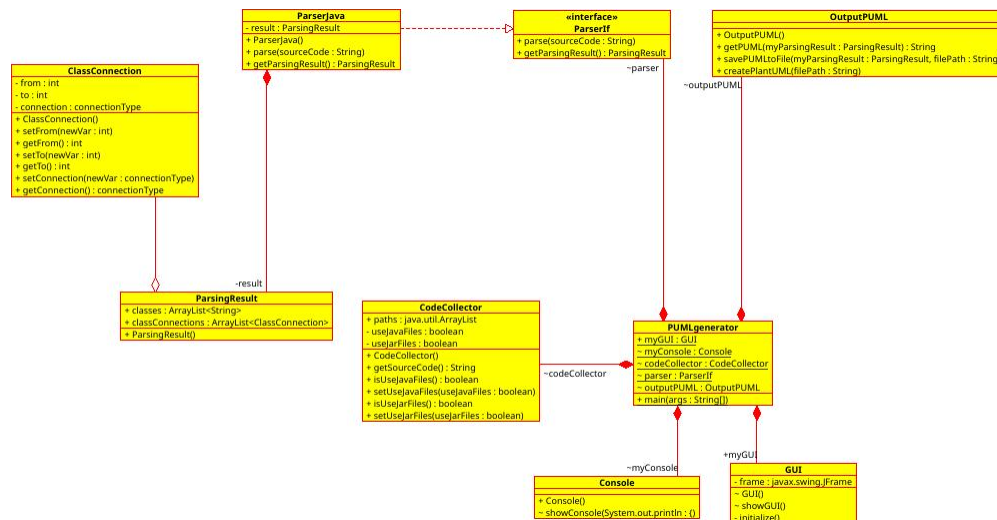


Abbildung 3: Klassendiagramm des Sprints

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

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

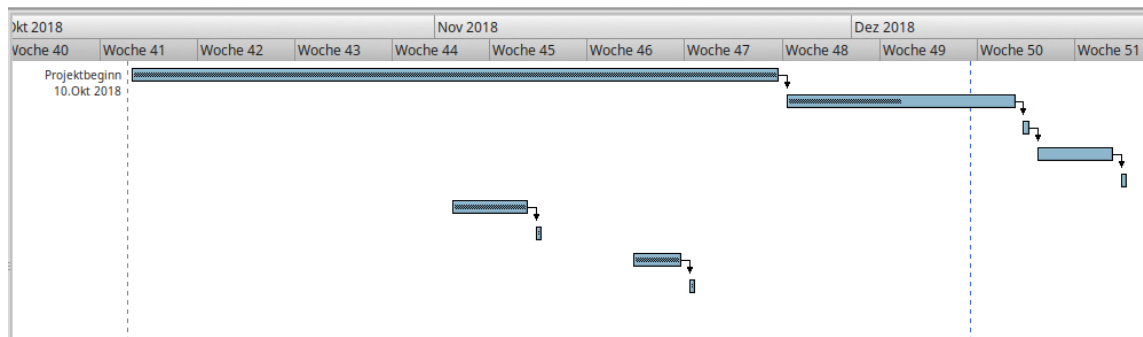


Abbildung 4: Gantt-Diagramm für Sprint 1

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" erstellen, in dem zusätzliche Testdateien landen

IV.10 Ergebnisse der Retrospektive

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

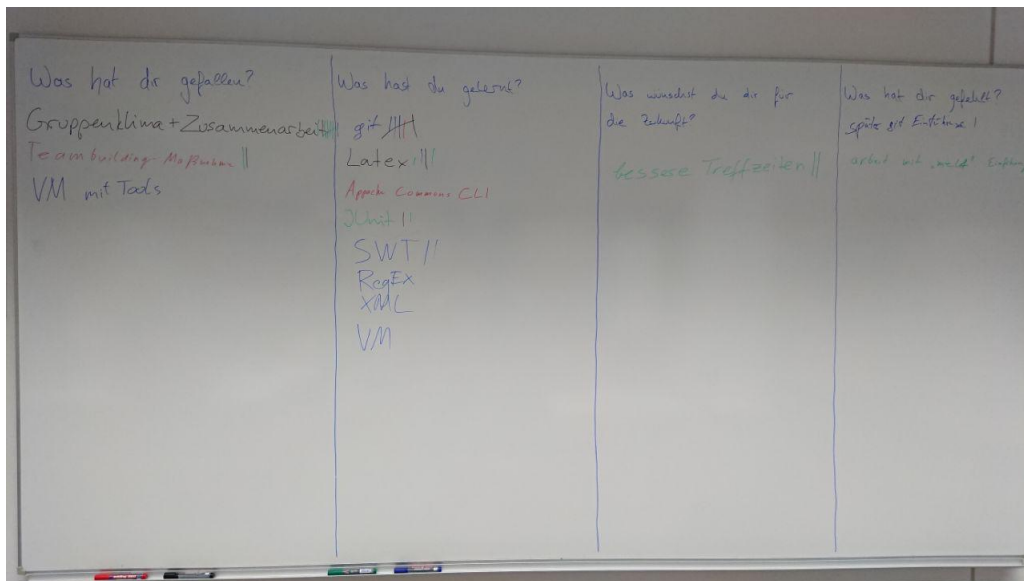


Abbildung 5: Anmerkungen der Teammitglieder zum bisherigen Verlauf des Projekts

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: Kontroll-Diagramm für Sprint 2

V.4 Liste der durchgeführten Meetings

- Planning-Meeting (21.02.2019)
- Zwischen-Meeting (08.04.2019)

- Zwischen-Meeting (11.04.2019)
- Review-Meeting (15.04.2019)

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 |
|----------------|--------------------|------------|------------|------|--------------------|
| Testdaten | Marian Geissler | 06.04.2019 | 15.04.2019 | 2 | |
| Logger | Patrick Otte | 08.02.2019 | 08.04.2019 | 12 | LogMain.java |
| Output | Patrick Otte | 11.04.2019 | 15.04.2019 | 3 | OutputPUML.java |
| Konsole | Johann Gerhardt | 14.04.2019 | 14.04.2019 | 1 | Console.java |
| Java-Parser | Michael Lux | 30.03.2019 | 30.03.2019 | 14 | ParserJava.java |
| GUI | Jan Sollmann | 01.04.2019 | 06.04.2019 | 5 | GUI_SWT.java |
| GUI | Julian Uebe | 21.02.2019 | 15.04.2019 | 7 | SWT-Tool |
| Code-Collector | Elisabeth Schuster | 07.02.2019 | 14.04.2019 | 5.5 | CodeCollector.java |
| Profiler | Elisabeth Schuster | 10.04.2019 | 26.04.2019 | 7 | Profiler |
| Java-Parser | Jona Meyer | 30.03.2019 | 30.03.2019 | 7 | ParserJave.java |
| Code-Collector | Leo Rauschke | 07.04.2019 | 14.04.2019 | 5.75 | CodeCollector.java |
| Profiler | Leo Rauschke | 09.04.2019 | 29.04.2019 | 3 | Profiler |
| Output | Tore Arndt | 11.04.2019 | 15.04.2019 | 5 | OutputPUML.java |

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 |
|-----------------|-------------------------|--|
| Testdatensatz | komplett | zukünftig als automatischer Ausgabe-Test |
| GUI_SWT.java | komplett | wird durch Swing-GUI ersetzt |
| CodeCollector | Pfadbehandlung | Funktioniert nun auch unter Windows |
| CodeCollector | einlesen | Funktioniert |
| ParserJava.java | buildTree | Es bestehen weitherhin Bugs |
| Alle | komplett | Unit-Tests für das ganze Programm folgen |
| ParserJava.java | buildTree | Erweiterung für das Erstellen von Sequenzdiagrammen |
| GUI_SWT.java | createContents, runPUML | Ausgabe für Sequenzdiagramme muss implementiert werden |

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.

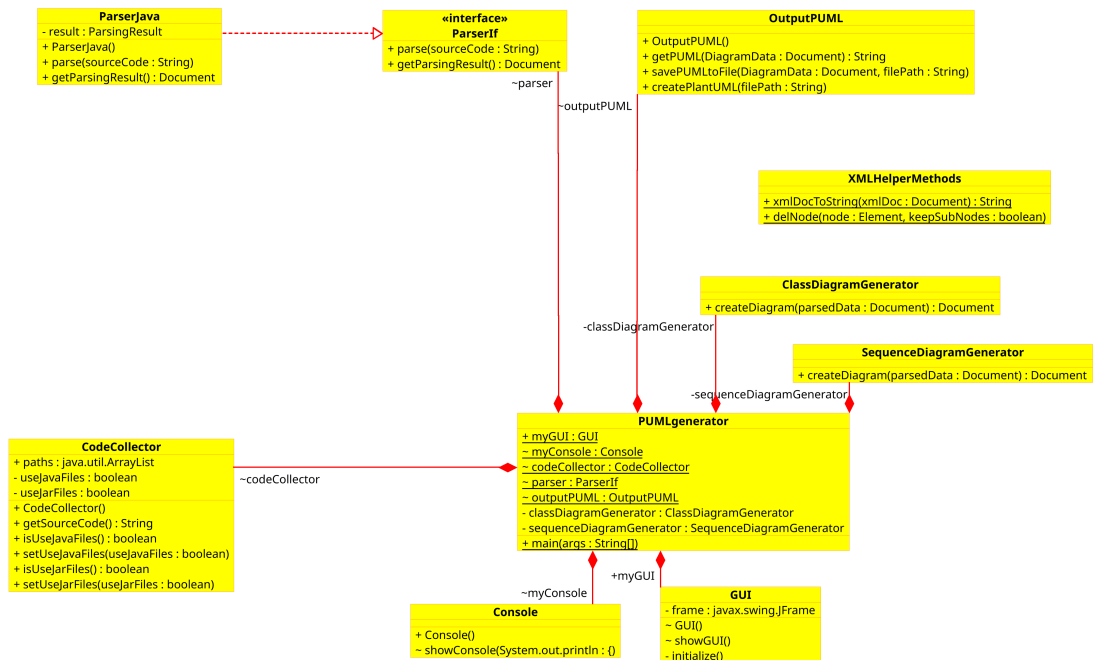


Abbildung 7: Klassendiagramm des Sprints



Abbildung 8: Kontroll-Diagramm für Sprint 3

VI.3 Kontroll-Diagramm

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 |
|--|--------------------|---------|---------|---------|-------------------------------|
| Profiler in Betrieb nehmen | Elisabeth Schuster | 26.4.19 | 26.4.19 | 1h | |
| BugfixParser | Michael Lux | 15.4.19 | 15.4.19 | 2h | ParserJava.java |
| BugfixParser | Jona Meyer | 28.4.19 | 29.4.19 | 10h | ParserJava.java |
| Unit-tests für den Gesammtaufruf | Marian Geißler | 17.4.19 | 28.4.19 | 7h | MainTest.java |
| Generator für Sequenzdiagramme - Spezifikation für xml-Datei | Leo Rauschke | 22.4.19 | 22.4.19 | 1h | SeqDiagram.xml |
| Generator für Sequenzdiagramme | Leo Rauschke | 18.4.19 | 18.4.19 | 3h 15m | SequenceDiagramGenerator.java |
| Generator für Sequenzdiagramme | Elisabeth Schuster | 22.4.29 | 24.4.19 | 3h 30m | SequenceDiagramGenerator.java |
| Übergabe der Klassendiagramme als XML | Tore Arndt | 21.4.19 | 24.4.19 | 25h | OutputPUML.java |
| Übergabe der Klassendiagramme als XML | Patrick Otte | 21.4.19 | 24.4.19 | 30h 15m | OutputPUML.java |
| GUI zu AWT/SWING umbauen | Julian Uebe | 15.4.19 | 16.4.19 | 8h | GUI_Swing.java |
| | | | | | |

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

VII.1 Ziel des Sprints

Ziel des Sprints ist, einen möglichst funktionalen Zwischenstand zu erreichen, der die Spezifikation erfüllt. Insbesondere Unit-Tests für die einzelnen Teile sowie die Weiterentwicklung der Sequenzdiagramm-Generierung sollen implementiert und die Umstrukturierung zu XML abgeschlossen werden. Ebenso soll die neue Rahmenstruktur umgesetzt werden.

VII.2 User-Stories des Sprint-Backlogs

VII.2.1 Klassen- und 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.

Erstellung von Klassendiagrammen Als Benutzer wünsche ich mir, Klassendiagramme aus meinem bestehenden Quellcode erstellen zu können, damit ich einen Überblick über die Klassen meines Programms und deren Beziehungen bekomme.

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

VII.3 Zeitliche Planung

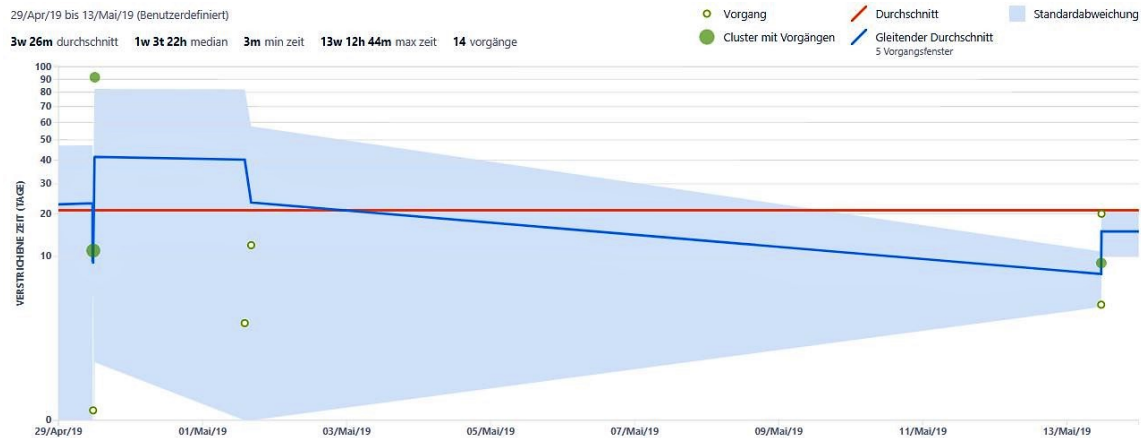


Abbildung 9: Zeit-Diagramm für Sprint 4

VII.4 Liste der durchgeführten Meetings

- Planning-Meeting (29.04.2019)
- Zwischenstandspräsentation (06.05.2019)
- Zwischen-Meeting (10.05.2019)
- Review-Meeting (13.05.2019)

VII.5 Ergebnisse des Planning-Meetings

Die XML- und Textspezifikationen sowie deren Transformationen sind allen Teams bekannt. Die Grundlage für die Weiterentwicklung der Sequenzdiagramm-Generierung ist damit gelegt.

VII.6 Aufgewendete Arbeitszeit pro Person+Arbeitspaket

| | | | | | |
|---------------------------------|-----------------|----------|----------|-----|---------------------------------|
| Unit-Tests für den Gesamtaufruf | Johann Gerhardt | 01.05.19 | 13.05.19 | 8 | MainTest.java |
| Neue Rahmenstruktur erstellen | Marian Geißler | 29.04.19 | 01.05.19 | 5.5 | Anpassungen an mehreren Klassen |
| GUI zu AWT/SWING umbauen | Julian Uebe | 29.04.19 | 13.05.19 | 8 | GUI_Swing.java |

| | | | | | |
|---|--------------------|----------|----------|-----|-----------------------------------|
| Generator für Sequenzdiagramme | Elisabeth Schuster | 29.04.19 | 01.05.19 | 8.5 | SequenzDiagramGenerator.java |
| Generator für Sequenzdiagramme | Leo Rauschke | 29.04.19 | 01.05.19 | 13 | SequenzDiagramGenerator.java |
| | | | | | SequenceDiagramGeneratorTest.java |
| Ausgabe für Sequenz- und Klassendiagramme | Tore Arndt | 07.05.19 | 13.05.19 | 20 | OutputPUML.java |
| Ausgabe für Sequenz- und Klassendiagramme | Patrick Otte | 29.04.19 | 13.05.19 | 15 | OutputPUML.java |
| | | | | | OutputPUMLTest_classdia.java |
| Generator für Klassendiagramme | Marian Geißler | 29.04.19 | 13.05.19 | 3.5 | ClassDiagramGenerator.java |
| | | | | | XmlHelperMethods.java |
| Generator für Klassendiagramme | Johann Gerhardt | 01.05.19 | 13.05.19 | 16 | ClassDiagramGenerator.java |
| Parser umbauen | Michael Lux | 08.05.19 | 13.05.19 | 14 | ParserJava.java |
| Parser umbauen | Jona Meyer | 08.05.19 | 13.05.19 | 10 | ParserJava.java |

VII.7 Konkrete Code-Qualität im Sprint

Aufgrund der Umstrukturierung sind einige Funktionalitäten noch nicht gegeben. Das Durchlaufen der XML-Bäume sollte teilweise noch angepasst werden. Hierfür bietet sich die Verwendung von XPath-Ausdrücken an.

VII.8 Konkrete Test-Überdeckung im Sprint

Es fehlen noch viele Unit-Tests und die vorhandenen sollten ausführlicher gestaltet werden. Ein Gesamttest für das Programm wurde geschrieben und hat eine hohe Testabdeckung.

VII.9 Ergebnisse des Reviews

| Klasse | Methode | Anmerkungen |
|------------------------------|---------------|---|
| GUI_Swing.java | allgemein | Anpassungen |
| SequenzDiagramGenerator.java | Instanzen | Vervollständigung bzgl. Spezifikation + Bugfixing |
| ClassDiagramGenerator.java | createDiagram | Vervollständigung bzgl. Spezifikation + Bugfixing |
| OutputPUML.java | getPUML | Output für Sequenzdiagramme anpassen |
| XmlHelperMethods.java | allgemein | weitere nützliche Methoden in die Klasse überführen |
| ParseJava.java | allgemein | Vervollständigung bzgl. Spezifikation + Bugfixing |

Sonstiges:

- XPath nutzen
- Rekursive Implementierung beim Abarbeiten von verschachtelten Elementen
- Fehlende Unit-Tests nachholen

- Evtl. in späterem Sprint: Handling von Methoden in Bedingungen

VII.10 Abschließende Einschätzung des Product-Owners

VII.11 Abschließende Einschätzung des Software-Architekten

XXX

VII.12 Abschließende Einschätzung des Team-Managers

VIII. SPRINT 6

VIII.1 Ziel des Sprints

Ziel des Sprints ist es unter anderem alte Codefragmente zu entfernen, einige Klassen auszubessern und an neue Voraussetzungen anzupassen und mit dem Erstellen des C++ Parsers zu beginnen. Vor allem sollte am Ende des Sprints das Programm lauffähig sein.

VIII.2 User-Stories des Sprint-Backlogs

VIII.2.1 Vorschau

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

VIII.2.2 Klassendiagramme

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

VIII.2.3 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.

VIII.3 Zeitliche Planung

Der Sprint geht vom 24.05.19 bis zum 07.06.19.

VIII.4 Liste der durchgeführten Meetings

- Planning-Meeting (24.05.2019)

VIII.5 Ergebnisse des Planning-Meetings

- Bugfixes und Feinschliff Sequenzdiagrammgenerator [PUML-92]
- XML Compare für UnitTests [PUML-93]
- GUI Auflistung Klassen/Methoden [PUML-94]
- C++ Parser schreiben [PUML-95]
- Sequenzdiagramm Output fertigstellen [PUML-96]

- Test openjdk oracle jdk [PUML-98]
- alte Codefragmente entfernen [PUML-84]
- Anpassung Konsole auf neue Methoden [PUML-85]
- Parser umbauen [PUML-89/90]
- alten Code entfernen
- PUML Logo bestimmt

VIII.6 Aufgewendete Arbeitszeit pro Person+Arbeitspaket

| | | | | | |
|--|--------------------|----------|----------|--------------|------------------------------|
| Parser umbauen | Michael Lux | 24.05.19 | 07.06.19 | 1w 4d 3h | ParserJava.java |
| Parser umbauen | Jona Meyer | 24.05.19 | 07.06.19 | 4d 6h | ParserJava.java |
| Bugfix und Feinschliff von Sequenzdiagrammen | Elisabeth Schuster | 24.05.19 | 07.06.19 | 1d 6h 15m | SequenzDiagramGenerator.java |
| SeqDiagram1 | Leonie Rauschke | 24.05.19 | 07.06.19 | 5h 50m | SequenzDiagramGenerator.java |
| XML-Compare für Unit-Tests | Patrick Otte | 24.05.19 | 05.06.19 | 2d | |
| Logger Logdatei | Patrick Otte | 24.05.19 | 03.06.19 | 3h | |
| getList Methode | Patrick Otte | 26.05.19 | 27.05.19 | 6h | |
| Anpassung Console auf neue Methoden | Marian Geißler | 24.05.19 | 07.06.19 | 1d 6h 30m | |
| GUI auflistung der Klassen und Methoden | Julian Uebe | 24.05.19 | 07.06.19 | 1w 7h | |
| C++-Parser erstellen | Jan Sollmann | 24.05.19 | 07.06.19 | 1w 3d 1h 30m | ParserCPP.java |
| C++-Parser erstellen | Johann Gerhardt | 24.05.19 | 07.06.19 | 4d 1h | ParserCPP.java |
| Sequenzdiagramm-Output fertigstellen | Tore Arndt | 24.05.19 | 07.06.19 | 2d 4h | |
| Test für Sequenzdiagramm Output schreiben | Patrick Otte | 24.05.19 | 27.05.19 | 4h | |

VIII.7 Konkrete Code-Qualität im Sprint

Alte und unbenötigte Codefragmente wurden entfernt. In einigen Klassen wurden Bugs behoben. Die Code-Qualität hat sich daher wahrscheinlich verbessert.

VIII.8 Konkrete Test-Überdeckung im Sprint

Es wurde eine Methode für das Vergleichen von XML-Dokumenten geschrieben. Mit dieser lassen sich beispielsweise der ClassDiagramGenerator und der SequenzDiagramGenerator testen. Daher sind in diesem Sprint neue Tests hinzugekommen, wodurch die Testabdeckung hoch gehalten wird.

VIII.9 Ergebnisse des Reviews

| Klasse | Methode | Anmerkungen |
|--------|---------|-------------|
|--------|---------|-------------|

VIII.10 Ergebnisse der Retrospektive

Der Sprint lief insgesamt betrachtet erfolgreich. Allerdings wird der CPP-Parser wohl nicht die gleiche Funktionalität wie der Java-Parser erreichen. Das Parsen von CPP-Quellcode funktioniert zwar in Ansätzen, ist allerdings weder vollständig noch für komplexeren Code ausgelegt.

VIII.11 Abschließende Einschätzung des Product-Owners

XXX

VIII.12 Abschließende Einschätzung des Software-Architekten

XXX

VIII.13 Abschließende Einschätzung des Team-Managers

XXX

IX. SPRINT 7

IX.1 Ziel des Sprints

Ziel des Sprints ist, ein funktionaler Zustand der Software. Die Bedienung soll möglichst fehlerfrei über GUI und Kommandozeile ermöglicht werden. Des weiteren sollen alte Codefragmente entfernt und die Struktur der einzelnen Klassen weiterhin verbessert werden. Da ein Ende des Projektes langsam absehbar ist, sollen Unit-Tests für eine größere Testüberdeckung sorgen, die einzelnen Teile sowie die Weiterentwicklung der Sequenzdiagramm-Generierung (Exceptions visualisiert) sollen angepasst werden, jedoch keine weitere Grundlegende Funktionalität hinzugefügt werden. Zusätzlich werden weitere Methoden in der XML-Hilfsklasse implementiert und für bessere Fehlerbehandlung der Logger erweitert. Weitere Auswahloptionen für eine verbesserte Nutzerinteraktion sollen in GUI und Konsole implementiert, sowie kleinere Bugfixes vorgenommen werden.

IX.2 User-Stories des Sprint-Backlogs

IX.2.1 Exceptions als Sequenzdiagramme

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

IX.2.2 Klassendiagramme

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

IX.2.3 Klassenauswahl

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

IX.2.4 Kommandozeile

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

IX.2.5 Methoden- und Variablenauswahl

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

IX.2.6 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.

IX.2.7 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.

IX.3 Zeitliche Planung

Für die Zeitliche Planung wurde von der ursprünglichen Idee, den Sprint eine Woche dauern zu lassen abgesehen, da ein Feiertag auf das Datum der Zwischenstandspräsentationen fiel. Somit wurde der Sprint 7 vom 07.06.2019 bis zum 17.06.2019 angesetzt.

IX.4 Liste der durchgeführten Meetings

- Planning-Meeting (07.07.2019)
- Meeting zur Präsentation des Zwischenstands (14.06.2019)
- Review-Meeting (17.06.2019)

IX.5 Ergebnisse des Planning-Meetings

Neben der Vereinbarung des angestrebten Funktionsumfangs, wurden weitere Bedingungen für Teile der Software zusammengetragen. Damit wurde die Spezifikation wie folgt erweitert:

- Variablen der Basisdatentypen eingefügt
- Zugriffsmodifikatoren für Instanzen, Variablen und Methoden eingefügt
- Konstruktor ist nun auch eine Methode in parsedData.xml
- static-modifier sollte nun geparsed und in Klassendiagramme übernommen werden
- Alle Methoden die kein Konstruktor sind, haben nun einen Result-Wert (ggf. void)
- Eigene parsedData.xml für c++

Außerdem soll die Software nun auch bei komplexeren Klassendiagrammen ein zuverlässiges Ergebnis liefern. Bezüglich der Sequenzdiagramme soll die Grundfunktionalität erreicht werden, verschachtelte Aufrufe, wie bspw. `instanz.methode().methode()` oder `instanz.methode(instanz.methode)` sollen vorerst nicht berücksichtigt werden, jedoch nicht zum Absturz des Programms führen.

Die Entwicklung des C++ Parsers wird vorerst auf Klassendiagramme eingeschränkt.

Das Anzeigen und Auswählen der Klassen fürs Klassendiagramm in GUI und Konsole sollte funktionieren, sowie das Anzeigen und Auswählen der Eintrittsmethode für das Sequenzdiagramm. Die Klasse "ClassDiagrammgenerator" soll über die boolischen Variablen: `showInstances`, `showVars` und `showMethods` verfügen und von außen gesetzt werden können. Die Grundlage für die Weiterentwicklung und das Erreichen eines soliden Funktionsumfangs der Software wurde damit gelegt.

IX.6 Aufgewendete Arbeitszeit pro Person+Arbeitspaket

| Arbeitspaket | Person | Start | Ende | h | Artefakt |
|-----------------|--------------|------------|------------|----|-------------------------------|
| Console | Marian G. | 07.06.2019 | 17.06.2019 | 23 | Console.java |
| Sequenzdiagramm | Tore A. | 07.06.2019 | 17.06.2019 | 20 | SequenceDiagramGenerator.java |
| Sequenzdiagramm | Patrick O. | 07.06.2019 | 17.06.2019 | 4 | SequenceDiagramGenerator.java |
| Logger | Patrick O. | 07.06.2019 | 17.06.2019 | 1 | Logger.java |
| Hilfsklasse xml | Leo R. | 07.06.2019 | 17.06.2019 | 4 | XmlHelperMethods.java |
| Klassendiagramm | Johann G. | 07.06.2019 | 17.06.2019 | 29 | ClassDiagrammGenerator.java |
| Hilfsklasse xml | Patrick O. | 07.06.2019 | 17.06.2019 | 1 | XmlHelperMethods.java |
| C++ Parser | Jan S. | 07.06.2019 | 17.06.2019 | 8 | XmlHelperMethods.java |
| Java Parser | Jona M. | 07.06.2019 | 17.06.2019 | 16 | ParserJava.java |
| Java Parser | Michael L. | 07.06.2019 | 17.06.2019 | 13 | ParserJava.java |
| Unit Tests | Elisabeth S. | 07.06.2019 | 17.06.2019 | 14 | *GeneratorTest.java |
| Gui | Julian U. | 07.06.2019 | 17.06.2019 | 8 | GUISwing.java |

IX.7 Konkrete Code-Qualität im Sprint

Durch das Entfernen der Codefragmente, die in nahezu jeder Klasse vorgenommen wurden hat das Projekt an Übersichtlichkeit gewonnen. In einigen Klassen (Konsole bspw.) wurden komplett redundante Codezeilen durch effizientere Lösungen ersetzt und die Wartbarkeit erhöht.

IX.8 Konkrete Test-Überdeckung im Sprint

Test Abdeckung hat sich verbessert, es fehlen neue Unit-Tests für den aktuellen Gesamt Ablauf.

IX.9 Ergebnisse des Reviews

| Klasse | Methode | Anmerkungen |
|-----------------------|-----------------|--|
| XmlHelperMethods | delNode | Löschen eines Knotens aus xml-Dokument, wurde implementiert und getestet |
| XmlHelperMethods | writeToFile | Generiert XML Datei, wurde implementiert und getestet |
| XmlHelperMethods | compareXml | Rückgabewert wurde geändert |
| Sequenzdiagramm | createDiagram | Bugfixes, wurde getestet |
| Logger | LogMain | Auf xml-Struktur umgestellt, korrekt angepasst |
| Gui | showGui | Auflistung der Klassen und Methoden, fehlt |
| ParserJava | parse | Bugs wurden gefixt, weitere Anpassungen ausstehend |
| Console | interactiveMode | Anpassung auf neue Methoden, Dateiauswahl, Fehlerbehandlung Nutzereingabe, wurde implementiert und getestet. Redundante Zeilen entfernt, Klassenvariablen eingebunden. |
| ClassDiagramGenerator | createDiagram | Ausgabe von Variablen und Methoden erweitert, wurde implementiert. Boolesche Variablen showInstances, showVars, showMethods fehlen. |

Sonstiges:

- Testabdeckung weiter erhöhen
- Präsentation (Plakat, Demo) vorbereiten
- Dritte Zwischenstandspräsentation am 21.06.2019

IX.10 Abschließende Einschätzung des Product-Owners

IX.11 Abschließende Einschätzung des Software-Architekten

XXX

IX.12 Abschließende Einschätzung des Team-Managers

Allgemein kann Sprint 7 als erfolgreicher und produktiver Sprint gewertet werden. Mit 10 geschlossenen Tickets wurde die Entwicklung zu einer stabilen und lauffähigen Version weiter vorangetrieben. Die Bedienbarkeit der Software ist verbessert worden und die Erzeugung von Sequenzdiagrammen ist gegeben. Es bestehen weiterhin kleinere Fehler im Parser und in bei der Erzeugung von Klassendiagrammen. Mit dem nächsten Sprint sollten diese Bugs allerdings behoben werden und die Software wird im vollen Maße einsatzbereit sein. Positiv anzumerken ist die selbstständige Arbeitsweise aller Teilnehmer, sowie die Zuweisung von Tickets und Nutzung der zur Verfügung stehenden Kommunikationsmittel.

X. DOKUMENTATION

X.1 Handbuch

X.1.1 Kommandozeilenaufruf

Aufruf:

```
java -jar puml.jar [-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 | Beschreibung |
|-----------|-------------------------|--------------------------------|---|
| -c | | Konsolenaufruf | keine grafische Oberfläche |
| -i | inputPathes | Input | durch Strichpunkt getrennte Liste der Pfade zu den einzulesenden Dateien und Ordnern |
| -o | outputPath | Output | muss ein Pfad zu einem Ordner sein. Hier soll der plantUML-Code und das Diagramm erzeugt werden |
| -ijar | | Ignore Jar | .jar-Dateien werden ignoriert |
| -ijava | | Ignore Java | .java-Dateien werden ignoriert |
| -cc | | create Classdiagram | erzeugt ein Klassendiagramm |
| -cs -s | entryClass, entryMethod | create Sequencediagram show | erzeugt ein Sequenzdiagramm listet alle Klassen und Methoden auf |
| -int | | interactive | interaktiver Modus |

X.1.2 Installationsanleitung

Allgemein Folgende Schritte müssen durchgeführt werden:

- Java oder OpenJDK in Version 11 installieren
- Graphviz installieren
- jar-Datei entpacken

Linux (Ubuntu) In Kommandozeile ausführen:

- `sudo apt-get install openjdk-11-jre graphviz unzip`
- `unzip puml_install.zip -d ~/myInstallPath`

Windows Es wird die Verwendung von OpenJDK-11 empfohlen.

- `puml_install.zip` entpacken
- OpenJDK-11 befindet sich in der zip-Datei. Installationsanleitung unter <https://stackoverflow.com/questions/52511778/how-to-install-openjdk-11-on-windows>
- Graphviz-installation durch doppelklick auf “graphviz-2.38.msi“ starten und durchführen

X.2 Software-Lizenz

X.2.1 Begründung der Entscheidung

Da die gesamte Ausgabe der Diagramme aus PUML auf der PlantUML Bibliothek basiert, welche selbst unter der GNU Public License (GPL) veröffentlicht wurde, kann PUML auch nur durch GPL lizenziert werden.

Die Grundregel ergibt sich aus Ziffer 2b) GPLv2:

You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

Somit fiel die Wahl der Lizenz für das PUML-Programm auf die GNU Public License.

X.2.2 Lizenztext

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

Dieses Programm ist Freie Software: Sie können es unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation, Version 3 der Lizenz oder (nach Ihrer Wahl) jeder neueren veröffentlichten Version, weiter verteilen und/oder modifizieren.

Dieses Programm wird in der Hoffnung bereitgestellt, dass es nützlich sein wird, jedoch OHNE JEDE GEWÄHR,; sogar ohne die implizite Gewähr der MARKTFÄHIGKEIT oder EIGNUNG FÜR EINEN BESTIMMTEN ZWECK. Siehe die GNU General Public License für weitere Einzelheiten.

Sie sollten eine Kopie der GNU General Public License zusammen mit diesem Programm erhalten haben. Wenn nicht, siehe <<https://www.gnu.org/licenses/>>.

XI. PROJEKTABSCHLUSS

XI.1 Protokoll der Abnahme und Inbetriebnahme beim Kunden

- Abnahme am 11.07.2019
- Kunde hatte Ubuntu16.04, daher musst OpenJDK von Hand nachinstalliert werden
- Einige Diagramme konnten zur Demonstration der Funktionalität erzeugt werden

XI.2 Präsentation auf der Messe

Poster, Bericht

XI.3 Abschließende Einschätzung durch Product-Owner

XXX

XI.4 Abschließende Einschätzung durch Software-Architekt

XI.4.1 Funktionalität

Die Kernfunktionalität wurde implementiert und funktioniert. Das Erzeugen der Klassendiagramme sollte für beliebigen Java-Code funktionieren. Das Erzeugen der Sequendiagramme funktioniert für die Java-Spezifikation. Hier fehlen noch folgende Sonderfälle:

- `instanz.function(instanz.function)`
- `instanz.functionMitInstAlsReturn.function()`

Das Erzeugen der Klassendiagramme für C++-Code funktioniert ebenfalls für die Spezifikation. Allerdings fehlen auch hier diverse Sonderfälle.

Die GUI ist soweit vollständig implementiert, hat aber noch einige kleine Bugs.

Die Kommandozeilen-Schnittstelle ist recht umfangreich.

XI.4.2 Tests

Die Testüberdeckung liegt bei ca. 62%. Wobei einige der Tests noch fehlschlagen. Größtenteils aber auch unwichtigen Gründen. Z.B. Vergleich von Bildern welche nicht Byte für Byte identisch sind.

XI.4.3 Architektur

Beim entwerfen der Architektur wurde darauf Wert gelegt, dass das Projekt modular aufgebaut ist. Dies vereinfachte zum einen die Verteilung der Aufgaben und ermöglicht zum anderen das einfache hinzufügen oder austauschen einzelner Module.

XI.4.4 Team

Das Team besteht wie bei Projektbeginn immer noch aus 10 Bachelor- und 2 Master-Studenten. Aufgrund der Größe des Teams fiel es uns als Projektleitung nicht immer leicht alle Teammitglieder mit Sinnvollen Aufgaben zu betrauen. Daher wurde auch viel im Pair-Programming gearbeitet. Dies funktionierte für die meisten Zweiergruppen sehr gut. Die Zusammenarbeit des Teams war

von Anfang an gut, verbesserte sich aber im Laufe des Projektes noch deutlich. Insbesondere wurden immer mehr für die Entwicklung relevante Informationen direkt unter den Teammitgliedern propagiert und nur noch für die Projektleitung wichtige Informationen an diese herangetragen. Das Team-Umfeld habe ich als professionell und angenehm empfunden. Zumal auch meist die Stimmung sehr gut war.

XI.4.5 Fähigkeiten der Teammitglieder

Während zu Beginn noch viele Teammitglieder mit den für sie neuen Technologien (Git, LaTeX, Linux als Betriebssystem, Eclipse als IDE, JUnit, ... später auch XML und XPath) zu kämpfen hatten, bereiteten diese gegen Ende des Projektes kaum mehr Probleme und es wurde sich stark auf die Weiterentwicklung des Quellcode konzentriert. Auch die Qualität des Quellcodes nahm beständig zu. Wobei hier aber noch durchaus Optimierungspotential besteht. Da man Programmieren aber meiner Ansicht nach größtenteils durch Praktische Erfahrung lernt und es Jahre braucht um es wirklich zu beherrschen, wäre alles andere Verwunderlich.

XI.5 Abschließende Einschätzung durch Team-Manager

Gibt es nicht.