Konzepte zur Bachelorarbeit

Template-basierte Synthese von
Verzweigungsstrukturen mittels L-Systemen

Adrian Helberg

9. November 2020

Inhaltsverzeichnis

1 Literaturübersicht

1.1 Softwaretechnik

Eine Fallstudie der Universität Karlsruhe [9] untersucht den Einsatz der Softwaretechnik **Extreme Programming** (XP) im Kontext der Erstellung von Abschlussarbeiten im Universitätsumfeld. Hierzu werden folgende Schlüsselpraktiken untersucht:

- XP als Softwaretechnik zur schrittweisen Annäherung an die Anforderungen eines Systems
- Änderung der Anforderungen an das Systems
- Funktionalitäten (**Features**) werden als Tätigkeiten des Benutzers (**User Stories**) definiert
- Zuerst werden Komponententests (Modultests) geschrieben und anschließend die Features (Test-driven Design)
- Keine seperaten Testing-Phasen
- Keine formalen Reviews oder Inspektionen
- Regelmäßige Integration von Änderungen
- Gemeinsame Implementierung (Pair Programming) in Zweiergruppen

Aus der Fallstudie geht hervor, dass Extreme Programming einige Vorteile bei der Bearbeitung eines Softwareprojektes einer Bachelorarbeit bietet. Zum einen können sich Anforderungen an das zu erstellende System durch parallele Literaturrecherche ändern, zum anderen können Arbeitspakete durch Releases abgebildet werden.

1.2 Grundlagen

- L-Systeme [8]
- Turtle-Algorithmus [12]
- Parametrisierte L-Syteme [13]
- String matching [10]
  - Verschiedene Algorithmen [5], [14]

1.3 Basisquelle

Bei *Inverse Procedural Modeling of Branching Structures by Inferring L-Systems*[4] geht es um ein Modell zum Lernen von L-Systemen von Verzweigungsstrukturen mithilfe maschinellen Lernens (*Deep Learning*) anhand beliebiger Grafiken. Hierzu werden atomare Strukturen mit einem neuronalen Netz erkannt, eine hierarchische Topologie (Baumstruktur) aufgebaut, aus der ein L-System inferiert und mit einem **Greedy** Algorithmus optimiert wird. Ausgabe des Systems ist ein generalisiertes L-System, aus dem ähnliche Strukturen, wie die der Inputgrafik, erstellt werden können.
Aus dieser Quelle werden folgende Konzepte genutzt:

- Nutzer einer Baumstruktur zur Organisation von genutzten atomaren Verzweigungsstrukturen (**Templates**) mit Knoten für Templates und Kanten für geometrische Transformationen
- Untersuchen der Baumstruktur auf Wiederholungen
- Parametrisierte L-Systeme (L-System mit **Modulen**) zur Abbildung von Transformationsparametern
- Kostenfunktion zur Bewertung eines L-Systems

2 Eigenleistung

Die Eigenleistung der Bachelorarbeit besteht aus:

- Algorithmus Baumstruktur $\rightarrow$ L-System
- Leere Symbole (ε) als Dummy-Knoten für Terminale
- Parameterverteilung als Histogramm
- Gewichtetes Randomisieren von Parametern

3 Dokumentation

3.1 Gliederung

Im Folgenden wird eine vorläufige Gliederung der schriftlichen Ausarbeitung gezeigt

1. Abbildungs- und Tabellenverzeichnis
2. Abkürzungsverzeichnis
3. Einleitung
   - 3.1. Problemstellung
   - 3.2. Ziele
   - 3.3. Methodik
   - 3.4. Aufbau
4. Grundlagen
   - 4.1. Grundbegriffe
   - 4.2. Grundlegende Arbeiten
   - 4.3. Verwandte Arbeiten
5. Konzepte
   - 5.1. Probleme & Lösungsansätze
   - 5.2. Architektur
   - 5.3. Algorithmen
6. Implementierung
7. Evaluierung
   - 7.1. Testumgebung
   - 7.2. Beobachtungen & Ergebnisse
   - 7.3. Diskussion und Bewertung
8. Ausblick
9. Literaturverzeichnis
10. Eidesstattliche Erklärung

4 Softwareprojekt

4.1 Vorgehen

Das Programm zu dieser Arbeit wird mit einem XP-basierten Ansatz erarbeitet. Hierbei beinhaltet ein **Release** Funktionen, die insgesamt für eine neue Version des Systems ausreichen; also ein vollständig funktionsfähiges Programm liefern. **User Stories** sind innerhalb der Iterationen umzusetzende Teilaufgaben und deren Aufwandseinschätzung gibt Auskunft über den Entwicklungsaufwand einer Umsetzung.

Umsetzung des Softwareprojektes in Iterationen mit folgenden Phasen:

- Planung:
  - Release-Planung:
    „*Welche Features werden in diesem Release umgesetzt?*“,
    User Stories, Aufwandsschätzung, Anforderungsmanagement
  - Iterationsplanung:
    Umwandlung der User Stories in kleine Arbeitsschritte,
    Festlegen der Dauer einer Implementierung
- Entwurf: Architektur, Klassendiagramme, Schnittstellen
- Testing: (Automatisierte) Modul- und Regressionstests
- Programmierung: Umsetzung der Features, Implementierung, Modularisierung

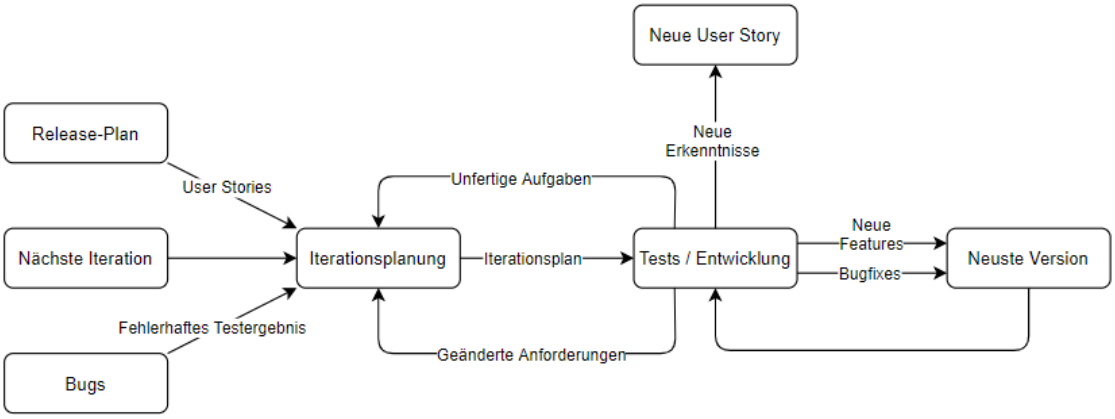

Abbildung 1: Ablaufdiagramm

4.2 Technnologien

- Programmiersprache: Java Version X mit
  - JavaFX Version X
- Build-Management-Tool: Gradle[3] Version X
- Versionskontrolle: Github Repository[2] via Git[1]
- IDE: JetBrains IntelliJ IDEA[7] 2020.2.2 (Ultimate Edition)
- Betriebssystem: Microsoft Windows 10 Pro 64 Bit
- Prozessor: Intel Core i5-3570K CPU @ 3.40GHz
- User Story Map: Trello Board[15]

5 Softwarearchitektur

Die Gliederung der Inhalte für die Softwarearchitektur erfolgt nach der arc42-Vorlage [6]

5.1 Einführung und Ziele

Ziel ist die Erstellung eines Programms zur Synthetisierung von Ähnlichkeitsabbildungen einer vom Benutzer erstellten Verzweigungsstruktur mittels Inferieren und Optimieren von L-Systemen. Die wesentlichen Features des Programms sind:

- Erstellung einer Verzweigungsstruktur über eine grafische Benutzeroberfläche (**GUI**)
- Einbindung atomarer Strukturen über externe Dateien
- Synthetisierung von ähnlichen Verzweigungsstrukturen anhand der erstellten Struktur
- Anzeigen von Verzweigungsstrukturen

Priorisierte (absteigend) Qualitätsziele, die bei der Erstellung des Systems umgesetzt werden sollten:

- **Funktionalität** durch Umsetzung aller Teilsysteme
- **Interoperabilität** durch Nutzen einer allgemeinen Repräsentation von L-Systemen, damit diese auch in anderen Programmen oder Algorithmen verwendet werden kann
- **Erweiterbarkeit** durch offene Entwurfsmuster (Design Pattern)
- **Modulare** Implementierung für effiziente Wartung und Erweiterung
- **Effizienz** durch effiziente Programmierung
- **Attraktivität** durch intuitive Benutzung (Benutzerfreundlichkeit)
- **Plattformunabhängigkeit** durch Verwenden des Java-Frameworks

5.2 Kontextabgrenzung

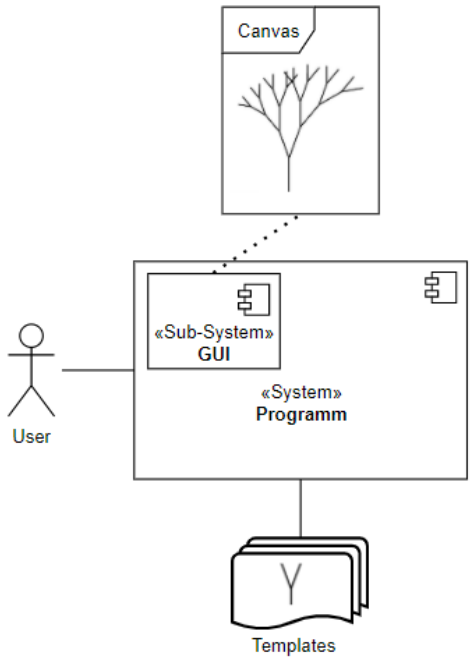

Abbildung 2: System und Systemumgebung

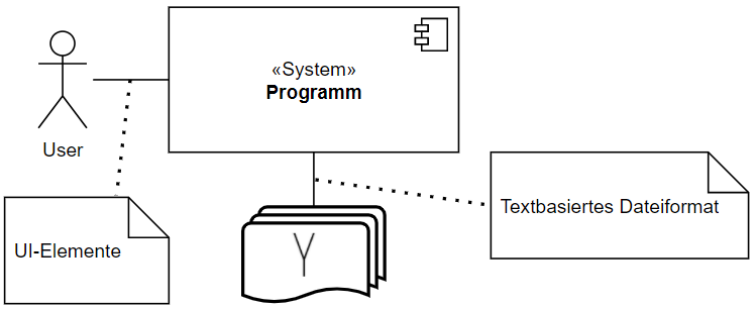

Abbildung 3: Technische Interaktion zwischen System und Systemumgebung

5.3 Lösungsstrategie

Gewählte Architekturansätze zur Erreichung der Qualitätsziele:

Qualitätsziel	**Architekturansatz**
Funktionalität	• Grafische Benutzerschnittstelle: *GUI* • Generieren der Baumstruktur: *TreeGenerator* • Ableiten von L-Systemen aus Baumstrukturen: *Inferer* • Generalisieren von L-Systemen: *Generalizer* • Randomisieren von L-System Parametern: *Randomizer*
Interoperabilität	Durch das Nutzen allgemeingültiger mathematischer Beschreibungen sollen erstellte L-Systeme in Fremdsystemen, wie Online Visualisierer, genutzt werden können
Erweiterbarkeit	Das Nutzen des **Pipeline Design Pattern**s soll das Erweitern des Systems durch Hinzufügen weiterer Teilschritte (**Pipes**) erleichtern. Trennung der grafischen Oberfläche und der Logik durch Aufbauen des Szenengraphen über ein XML-Dateiformat
Modularität	Sowohl eine sinnvolle Aufteilung von Funktionalitäten auf Dateien und Pakete (**Package**s), als auch effiziente Datenkapselung und geschlossene Informationskontexte sorgen für Modularität des Programms

Die Implementierung des Programms setzt sich aus folgenden Teilschritten zusammen:

- Erstellung der *GUI* (Paket gui, tree) mit
  - UI-Elementen
  - Render-Canvas
  - Dateianbindung der Templates
  - Erstellung der repräsentativen Baumstruktur (*treeGenerator*, Paket tree)
- Implementierung der Subsysteme als Pipes
  - *Inferer* (Paket grammar): Ableiten eines kompakten L-Systems aus einer Baumstruktur
  - *Generalizer* (Paket grammar): Generieren eines generalisierten L-Systems anhand eines „kleinen“ L-Systems
  - *Randomizer* (Paket grammar): Erzeugung von L-Systemen, die der erstellen Verzweigungsstruktur „ähnlich“ sind
- Komponenten- und Systemtests

Subsystem	**Umsetzung**
GUI	JavaFX als Framework zur Erstellung von grafischen und interaktiven Inhalten. Erstellung der Baumstruktur über dynamisches Erzeugen von Konten während der Strukturierung der Verzweigungsstruktur
Inferer	Algorithmus zum Iterieren maximaler Sub-Bäume und deren Reduzierung mittels Ersetzung durch Symbole und der zugehörigen Produktionsregel, bis eine Kostengrenze, die durch eine Kostenfunktion abgebildet werden kann, erreicht ist
Generalizer	
Randomizer	

5.4 Bausteinsicht

Ebene 1

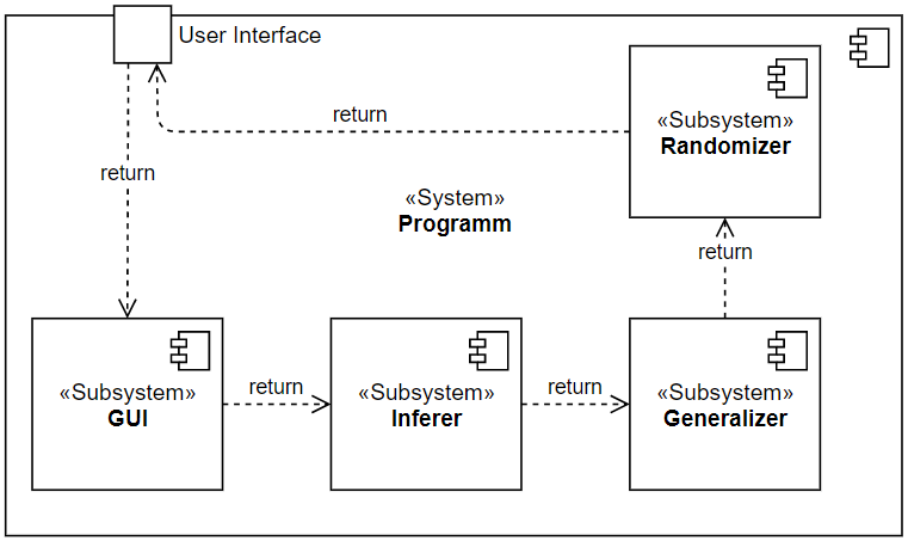

Abbildung 4: Subsysteme mit fachlichen Abhängigkeiten

Ebene 2

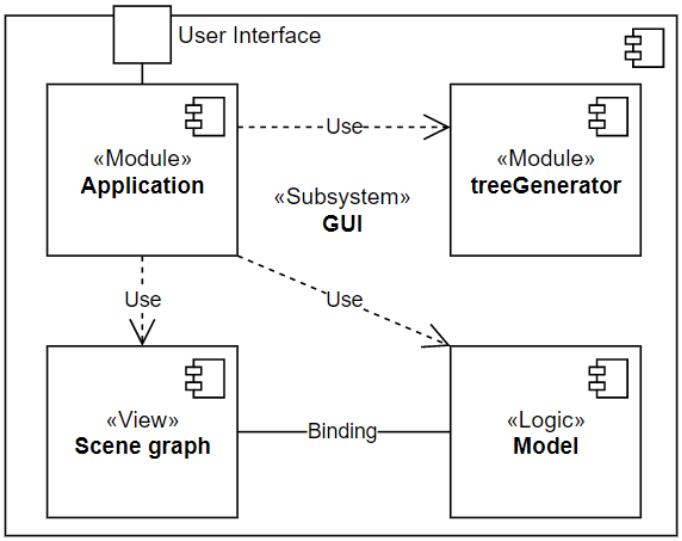

Abbildung 5: Subsystem GUI

5.5 Laufzeitsicht

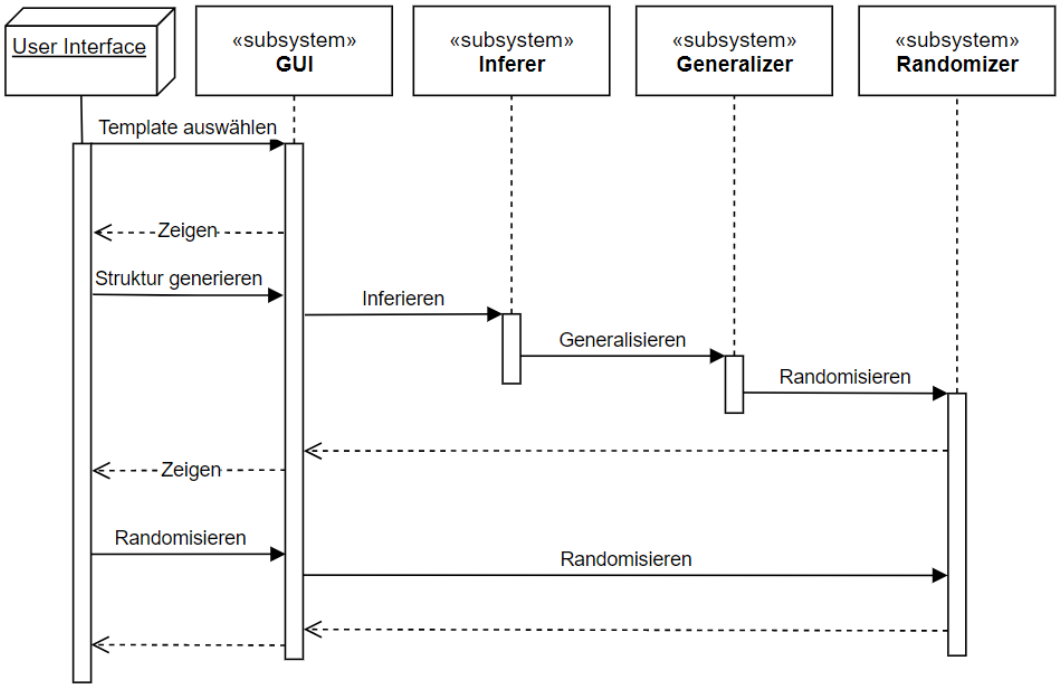

Abbildung 6: Laufzeitsicht

5.6 Verteilungsicht

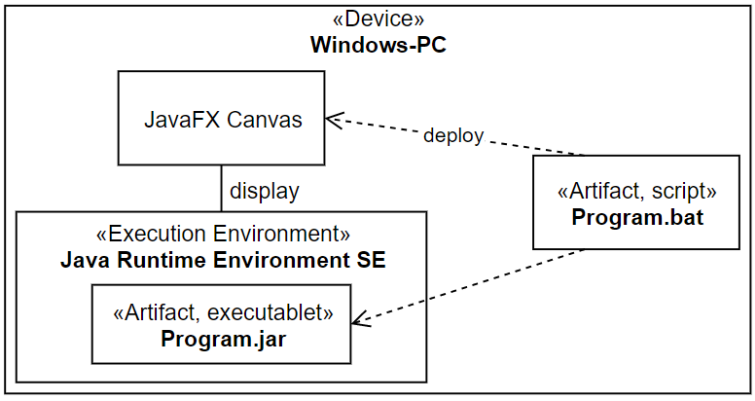

Abbildung 7: Infrastruktur Windows-PC

5.7 Konzepte (todo)

Testbarkeit

Validierung

Fehlerbehandlung

- Exception Handling
- Logging

Datenstrukturen

Workflows

Algorithmen

5.8 Entscheidungen (todo)

Risiken

Qualitätsmerkmale

Alternativen

Aufwand der Implementierung

6 Releases

Dieses Kapitel beschreibt die Release-Planung des Softwareprojekts im Sinne eines XP-orientierten Ansatzes:
Zuerst werden die Funktionalitäten, die im entsprechenden Release umgesetzt werden sollen definiert und **Epics** zugeteilt. Anschließend werden feingranulare User Stories formuliert, die mit einer Aufwandseinschätzung versehen werden. Mit diesen Informationen lässt sich dann die Iterationsplanung umsetzen. Die Umwandlung der User Stories in Arbeitsschritte (**Tasks**) und das Festlegen deren Dauer wird hier umgesetzt.

Beispiel: Release 1

Erstellung einer grafischen Benutzeroberfläche zur Erstellung von Verzweigungsstrukturen
Siehe Exposé Kapitel 1.1.2 Überblick Punkt I. Strukturieren und II. Visualisieren

6.1 Planung

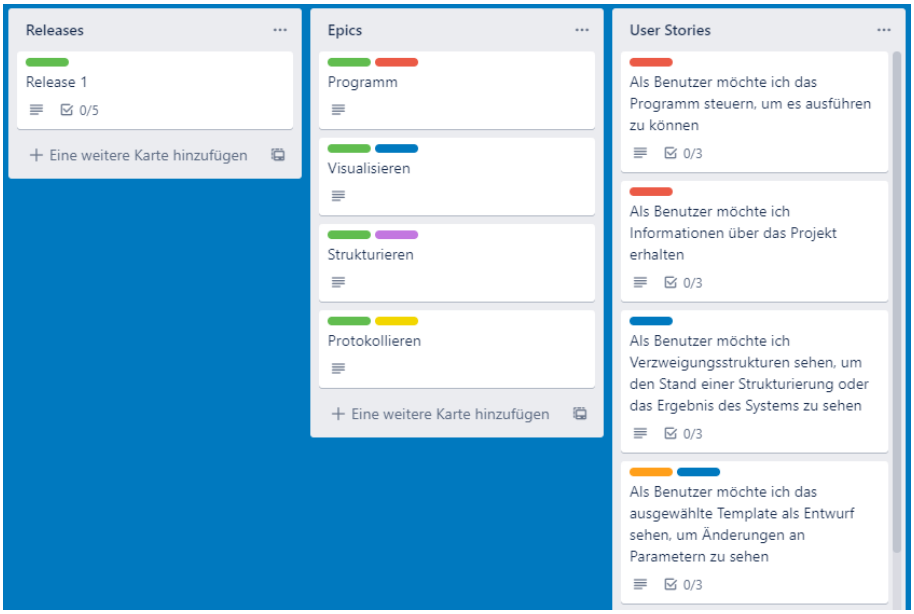

Abbildung 8: Release-Planung: User Story Map mit Releases, Epics und User Stories

Release 1 enthält 4 Epics wird also in 4 Iterationen erarbeitet:

- Iteration 1 (Epic Programm, 2 User Stories), Gesamtdauer: 6h

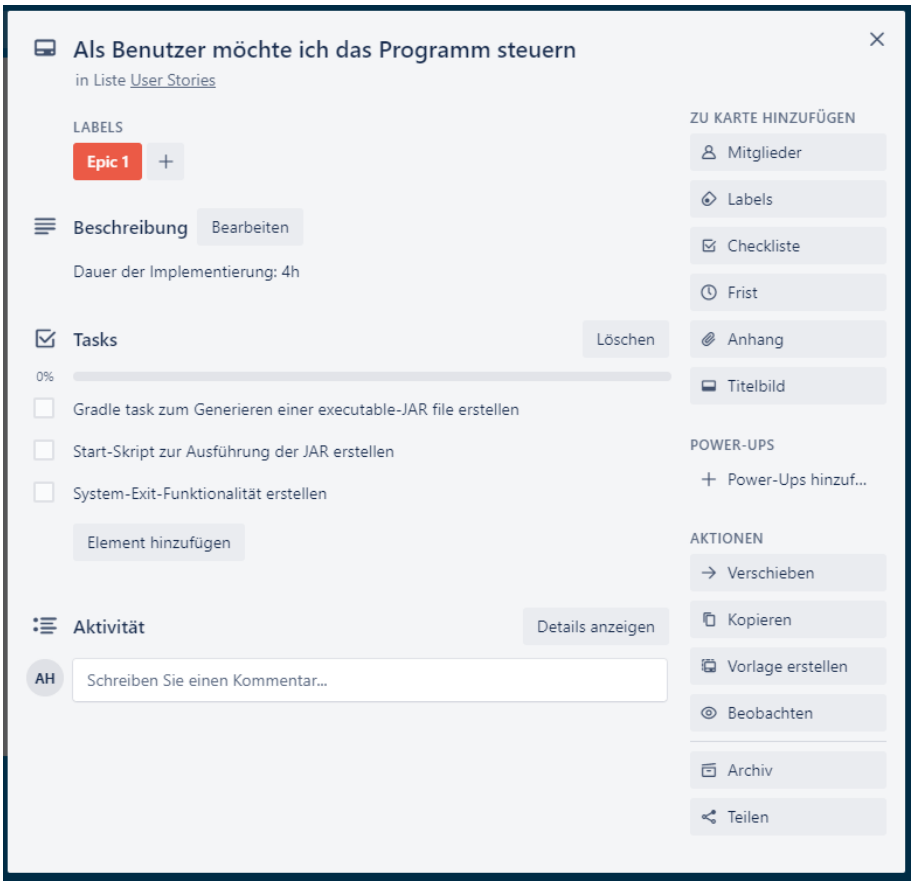

Abbildung 9: Iterationsplanung: User Story mit Tasks und Dauer

- Iteration 2 (Epic Visualisieren, 2 User Stories), Dauer: 18h
- Iteration 3 (Epic Strukturieren, 5 User Stories), Dauer: 18h
- Iteration 4 (Epic Protokollieren, 1 User Story), Dauer: 2h

6.2 Entwurf

Ein menschlicher Benutzer (**User**) nutzt das Programm, um eine Verzweigungsstruktur zu strukturieren. Hierzu wählt dieser einige Templates aus einer Sammlung vorgefertigter Templates aus, setzt Parameter und bestimmt den Ort der Platzierung. Die zu erstellende Struktur ist jeder Zeit sichtbar (Visualisierung).

Das System wird als Java-Programm mit main-Routine erstellt.

TODO

6.3 Testing

TODO

6.4 Programmierung

TODO

7 Implementation

Da die Arbeitspakete (Releases) iterativ und abgeschlossen erarbeitet werden, wird das **Pipeline** Design Pattern[11] genutzt.

Quellen

[1] Git - the stupid content tracker. `https://git-scm.com/`. Git is a member of Software Freedom Conservancy.

[2] Github. `https://github.com/adrian-helberg/bachelor`. GitHub, Inc. (2020).

[3] Gradle build tool. `https://gradle.org/`. Gradle Inc. 2020.

[4] Jianwei Guo, Haiyong Jiang, Bedrich Benes, Oliver Deussen, Xiaopeng Zhang, Dani Lischinski, and Hui Huang. Inverse procedural modeling of branching structures by inferring l-systems. *ACM Trans. Graph.*, 39(5), June 2020.

[5] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, June 1975.

[6] Dr. Peter Hruschka and Dr. Gernot Starke. arc42 softwarearchitektur-template. `https://arc42.de/template/`. Pragmatisches Muster für die Erstellung, Dokumentation und Kommunikation von Software- und Systemarchitekturen.

[7] Jetbrains intellij idea. `https://www.jetbrains.com/idea/`. Capable and Ergonomic IDE for JVM.

[8] A Lindenmayer. Mathematical models for cellular interactions in development. i. filaments with one-sided inputs. *Journal of theoretical biology*, 18(3):280—299, March 1968.

[9] M. M. Muller and W. F. Tichy. Case study: extreme programming in a university environment. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 537–544, 2001.

[10] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

[11] Pipeline design pattern. `https://java-design-patterns.com/patterns/pipeline/`. Datenverarbeitung in mehreren sequenziellen Schritten.

[12] P Prusinkiewicz. Graphical applications of l-systems. In *Proceedings on Graphics Interface '86/Vision Interface '86*, page 247–253, CAN, 1986. Canadian Information Processing Society.

[13] P. Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, Berlin, Heidelberg, 1990.

[14] Klaus Schulz and Stoyan Mihov. Fast string correction with levenshtein-automata. *INTERNATIONAL JOURNAL OF DOCUMENT ANALYSIS AND RECOGNITION*, 5:67–85, 2002.

[15] Trello board. `https://trello.com/`. Trello Board, 2020 Atlassian.