Inhaltsverzeichnis

1. Einführung **1**
- 1.1. Motivation . . . 2
- 1.2. Zielsetzung . . . 3
- 1.3. Aufbau der Arbeit . . . 3

2. Grundlagen **4**
- 2.1. Modellgetriebene Softwareentwicklung (MDSD) . . . 5
  - 2.1.1. Domäne . . . 5
    - 2.1.1.1. Definition . . . 5
    - 2.1.1.2. Domänenanalyse . . . 5
    - 2.1.1.3. Feature Modelling . . . 5
  - 2.1.2. Metamodell . . . 5
    - 2.1.2.1. Definition . . . 5
    - 2.1.2.2. Abstraktes Modell . . . 5
    - 2.1.2.3. Konkretes Modell . . . 5
  - 2.1.3. Domänenspezifische Sprache (DSL) . . . 5
    - 2.1.3.1. Definition . . . 5
    - 2.1.3.2. General Purpose Language (GPL) . . . 5
    - 2.1.3.3. Interne DSLs . . . 5
    - 2.1.3.4. Externe DSLs . . . 5
    - 2.1.3.5. Parser . . . 5
  - 2.1.4. Code Generator . . . 5
    - 2.1.4.1. Definition . . . 5
    - 2.1.4.2. Techniken zur Generierung von Code . . . 5
    - 2.1.4.3. Zusammenhang zu Transformatoren . . . 5
    - 2.1.4.4. Abgrenzung zum Compilerbau . . . 5
- 2.2. Software Architektur . . . 5
  - 2.2.1. Prinzipien der Softwaretechnik . . . 5
    - 2.2.1.1. Abstraktion . . . 5
    - 2.2.1.2. Bindung und Kopplung . . . 5
    - 2.2.1.3. Modularisierung . . . 5
    - 2.2.1.4. Weitere Prinzipien . . . 5
    - 2.2.1.5. Abhängigkeiten . . . 5
  - 2.2.2. Trennung durch Modelle . . . 5
  - 2.2.3. Modell-Transformations-Pipeline . . . 5

2.3. Design Pattern objektorientierter Programmierung 5
2.3.1. Visitor Pattern 5
2.3.1.1. Zweck 5
2.3.1.2. Beschreibung 5
2.3.1.3. Anwendbarkeit 5
2.3.2. Builder Pattern und Fluent Interfaces 5
2.3.2.1. Zweck 5
2.3.2.2. Beschreibung 5
2.3.2.3. Anwendbarkeit 5
2.3.3. Factory Pattern 5
2.3.3.1. Zweck 5
2.3.3.2. Beschreibung 5
2.3.3.3. Anwendbarkeit 5

3. Analyse **6**
3.1. Fragestellung: Wie kann, ausgehend von bestehendem Java-Code, ein Meta-Generator zur Erhöhung der Wirtschaftlichkeit Modellgetriebener Softwareentwicklung umgesetzt werden? 6
3.2. Aufwand eines konventionellen Softwareprojektes im Vergleich zu MDSD 6
3.3. Automatisierung der Entwicklung eines Code Generators 6
3.3.1. Java Code als Ausgangsmodell 6
3.3.1.1. Anreicherung des Java Codes mit Informationen 6
3.3.1.2. Parsen des Java Codes 6
3.3.2. Abstrakte Darstellung von Java Code als Modell 6
3.3.2.1. Anforderungen an das Modell 6
3.3.2.2. Schnittstellen zur Befüllung des Modells 6
3.3.3. Generierung von Java Code 7
3.3.3.1. Techniken zur Code Generierung 7
3.3.3.2. Vorhandene Frameworks zur Java Code Generierung . . 7

4. Konzept **8**
4.1. Einsparung der Implementation des Code Generators durch einen Meta-Generator 9
4.2. Bestandteile des Meta-Generators 9
4.2.1. Von Java-Code zum Annotations-Modell 9
4.2.1.1. Annotationen als Mittel zur Informationsanreicherung . 9
4.2.1.2. Parsen des annotierten Codes 9
4.2.1.3. Zweck des Annotations-Modells 9
4.2.2. Das CodeUnit-Modell 9
4.2.2.1. Baumstruktur des Modells 9
4.2.2.2. Spezialisierung durch ein Typ-Feld 9
4.2.2.3. Parametrisierung durch generische Datenstruktur 9

4.2.3. Generierung von Buildern als interne DSL aus dem Annotations-Modell . . . 9
4.2.3.1. Komposition der Builder aus benötigten Builder-Methoden 9
4.2.3.2. Übertagung vorgegebener Informationen in einen Builder 9
4.2.4. Erzeugung von Java Code aus einem befüllten CodeUnit-Modell . 9
4.2.4.1. Transformation des CodeUnit-Modells zum JavaFile-Modell 9
4.2.4.2. Erzeugung von Quelldateien . . . 9

5. Lösung: Spectrum (Proof of Concept) 10
5.1. Verwendete Bibliotheken . . . 11
5.2. Verwendeter Glossar . . . 11
5.2.1. JavaParser mit JavaSymbolSolver . . . 11
5.2.2. JavaPoet . . . 11
5.3. Architekturübersicht . . . 11
5.3.1. Amber . . . 11
5.3.1.1. Annotationen . . . 11
5.3.1.2. Parser . . . 11
5.3.1.3. Modell . . . 11
5.3.2. Cherry . . . 11
5.3.2.1. Generator . . . 11
5.3.2.2. Modell . . . 11
5.3.2.3. Generierte Builder . . . 11
5.3.3. Jade . . . 11
5.3.3.1. Transformator . . . 11
5.3.4. Scarlet . . . 11
5.3.4.1. Generator . . . 11
5.3.4.2. Modell . . . 11

6. Evaluierung 13
6.1. Kozept & Implementation . . . 13
6.1.1. Amber . . . 13
6.1.2. Cherry . . . 13
6.1.3. Jade . . . 13
6.1.4. Scarlet . . . 13
6.2. Softwarequalität . . . 13
6.2.1. Functionality . . . 13
6.2.2. Maintainability . . . 13
6.2.3. Performance . . . 13
6.2.4. Usability . . . 13
6.3. Grenzen des Lösungsansatzes . . . 13

7. Abschluss 14
7.1. Zusammenfassung . . . 14
7.2. Ausblick . . . 14

A. Dokumentation **15**
A.1. Verwendung der Annotationen 15
A.2. Verwendung der generierten CodeUnit-Builder 15
A.3. Klassendokumentation 15
A.3.1. Amber 15
A.3.2. Cherry 15
A.3.3. Jade 15
A.3.4. Scarlet 15

Verzeichnisse **16**

Literatur **18**

Eidesstattliche Erklärung **19**

Zustimmung zur Plagiatsüberprüfung **20**