UNIKASSEL
VERSITÄT

SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

Codegenerierung

Jens Kosiol

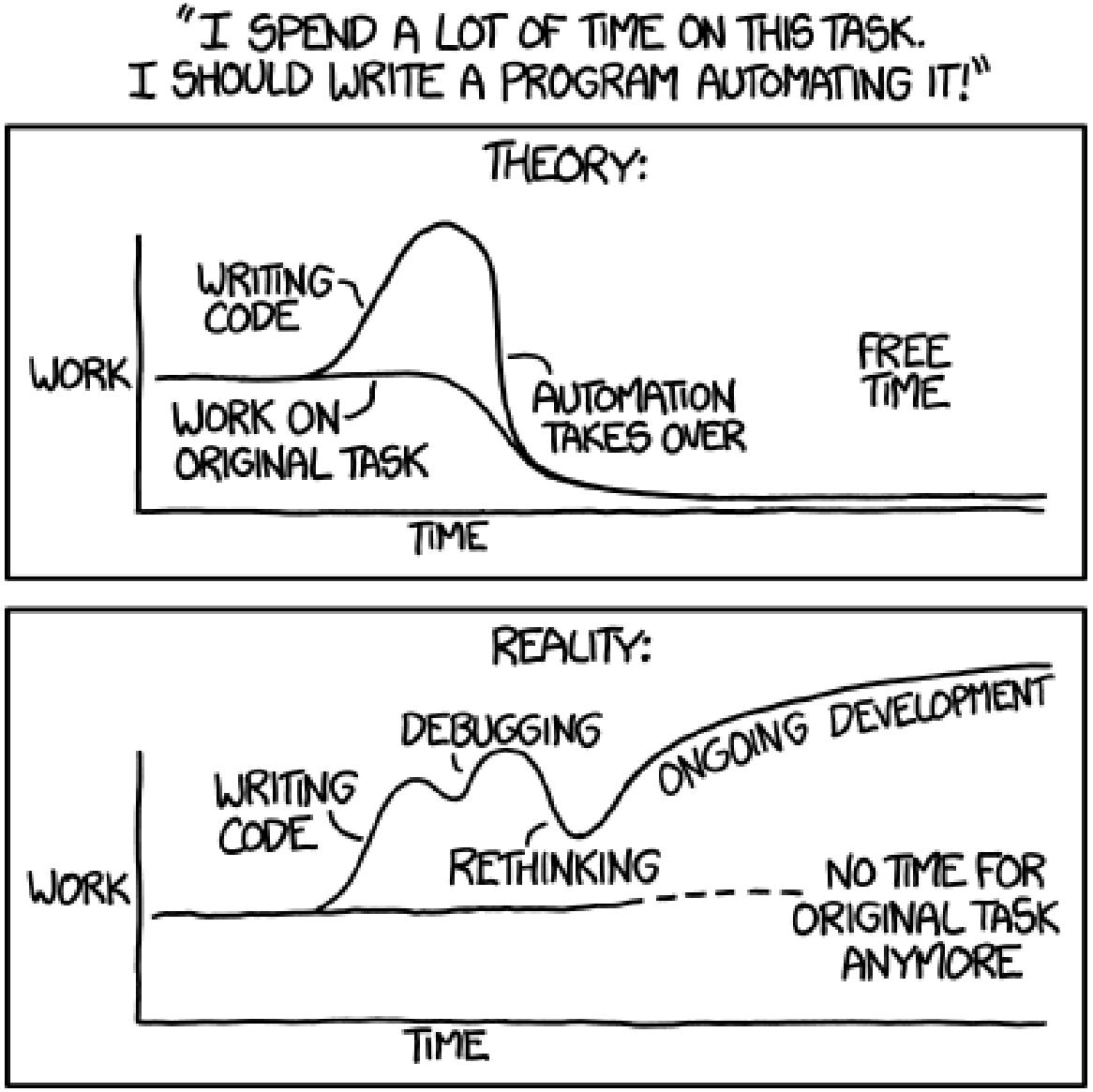
Wintersemester 23/24

Automatisierung

Als schematische Aufgabe kann das Erstellen von Code aus einem Datenmodell automatisiert werden.

Vorteile:

- Zeitersparnis und Vermeidung langweiliger, repetitiver Aufgaben
- Vermeiden von Fehlern/Garantie von Korrektheit
- Uniformität in Umsetzung
- Einhaltung von Codingstandards
- Adressierung verschiedener Zielsprachen, -platformen, …
- …



[Quelle: https://xkcd.com/1319/]

Ansätze für die Codegenerierung

Es gibt viele Ansätze für die Generierung von Code:

- Generierung von Code aus natürlichsprachlicher Anforderungsbeschreibung per Machine Learning (LLMs, NLP)
- Code-Templates für wiederkehrende Strukturen (Boilerplate Code)
- Domänenspezifische (Modellierungs-)Sprachen [DS(M)Ls]: Strukturierte (u.U. graphische) Sprache zum Beschreiben einer Domäne, die zu Code übersetzt werden kann
- …

Generierung von Code aus Datenmodell

Es gibt viele Tools, die Codegenerierung aus einem Datenmodell ermöglichen, mit unterschiedlichen Ansätzen und Funktionen.

- Generierung aus graphischen oder textuellen Darstellungen des Datenmodells
- Generierung von Code für Datenmodell (plus Boilerplate-Code wie Getter/Setter) oder auch teilweise Generierung von Programmlogik (durch Unterstützung für weitere Modellarten wie State Charts)
- Viele Tools aus dem UML Umfeld
  - Umple, MontiCore, Eclipse Modeling Framework, …

Fulib.org

Toolsuite für Softwareentwicklung und Lehre

- Entwickelt am Fachgebiet Software Engineering der Uni Kassel seit 2019: https://fulib.org/
- Entwickelt Fujaba und SDMLib weiter
- Einsetzbar in eigener, browserbasierter IDE oder als über Gradle eingebundene Library
- Funktionalitäten, die wir nutzen:
  - Codegenerierung aus Datenmodell (Datenmodell in Java-interner, textueller DSL beschrieben)
  - Generierung von Diagrammen

Grundsätzliches Vorgehen in Fulib

1. Fulib über Gradle als Dependency einbinden
2. Klassenmodell textuell beschreiben
   - Pfad: `src/gen/java`
   - Die `decorate`-Methode ist für die eigentliche Generierung zuständig
3. Ausführen der `decorate`-Methode durch Ausführen von `gradle generateScenarioSource`
4. Generierter Code
   - Pfad: `src/main/java`
   - Generiert die beschrieben Klassen inklusive Feldern, Referenzen, Gettern und Settern, die referentielle Integrität sicherstellen, und Boilerplate-Code für Event-Handling

Dokumentation unter https://fulib.org/docs/fulib/README.md

Grundlegende Semantik

- Die Klassen, die generiert werden sollen, werden als innere Klassen einer `GenModel`-Klasse deklariert, die das Interface `ClassModelDecorator` implementiert.
- Attribute werden (ohne Modifikator für die Sichtbarkeit) ganz normal deklariert.
- Assoziationen werden mit `@Link` annotiert und dann wie gewohnt deklariert. Argument der Annotation: Der Name (als `String`) der Assoziation **in Gegenrichtung**.
- Die `decorate`-Methode des `ClassModelDecorator` muss überschrieben werden; Aufruf von `haveNestedClasses(GenModel.class)` zur Generierung des Codes