technische universität
dortmund

Software- Engineering für langlebige Systeme

Thomas Ruhroth | Dortmund
SoSe 2014

Reengineering

–

Aus Alt mach Neu

VL8

- Problem des Reengineerings
- Übersicht über die Themenbereich

- Ziele:
  - Verstehen was Reengineering ist und in welche Bereiche angeschnitten werden

- Die meisten Blöcke die hier vorkommen werden in andern Vorlesungen der TU genauer behandelt.
- Beispiel: Prozess-Mining in MGSE

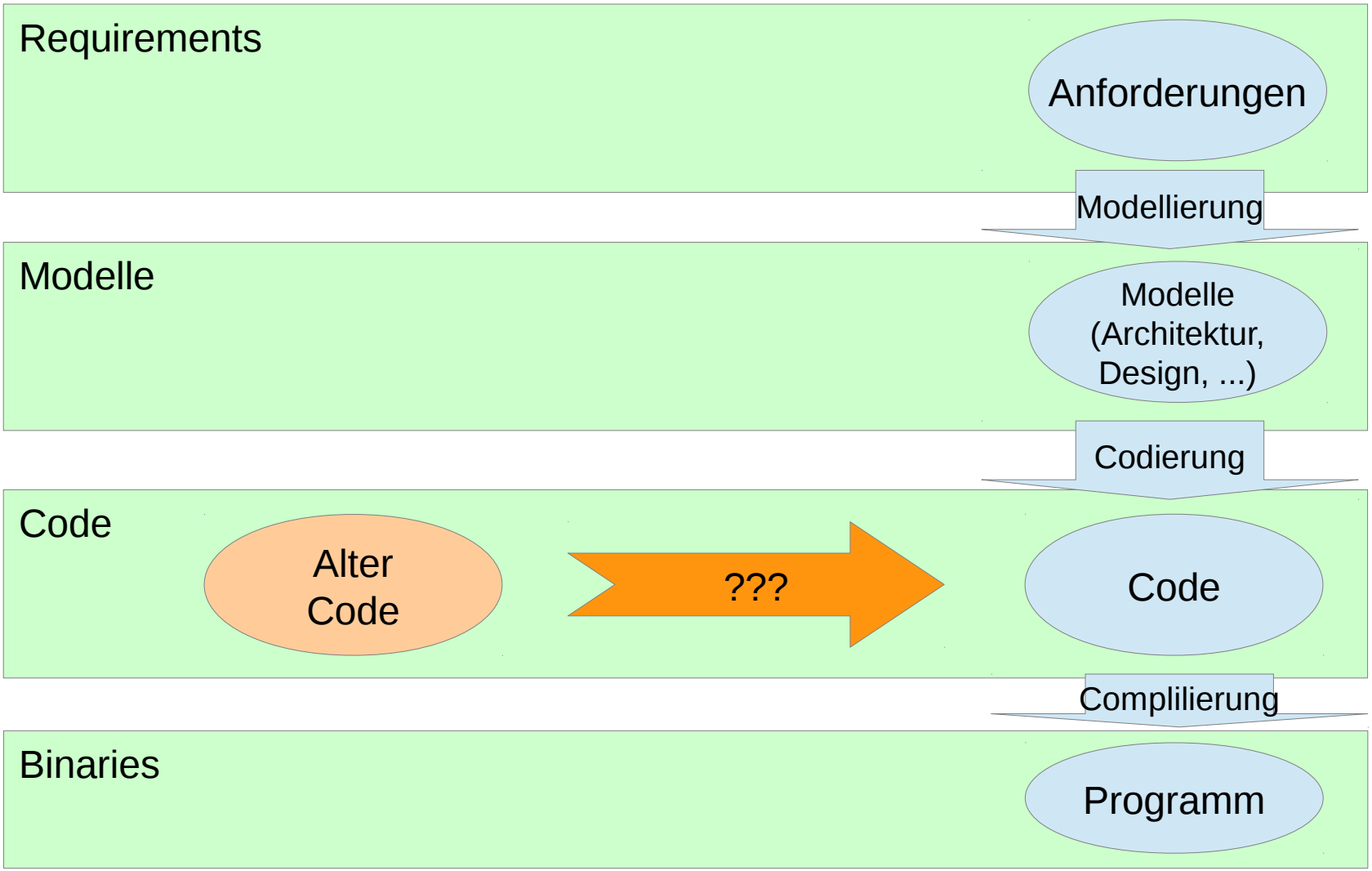
Problem

- Ein altes Programm muss angepasst werden
  - Systemwechsel
  - Änderungen der Businessprozesse bei Code ohne Modellen
  - Binaries ohne Code
- Nur Binaries/Code vorhanden
- Änderungen umfangreich

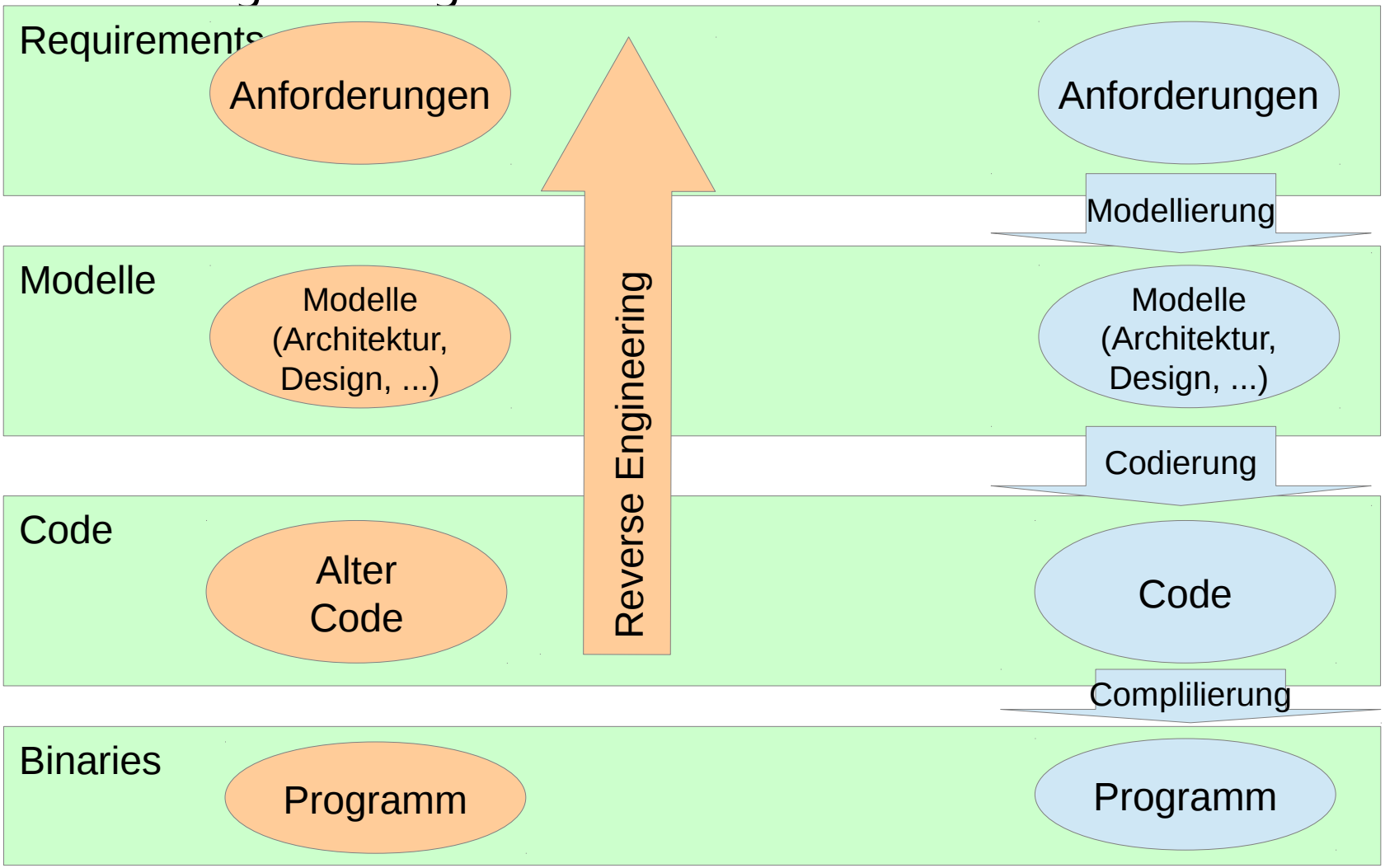
- Model-Dokumente nicht lesbar/verloren
  - Datenverluste
  - Nicht mehr lesbare Formate (siehe auch Folien zur Dokumentation)

Erinnerung Model-Driven-Development

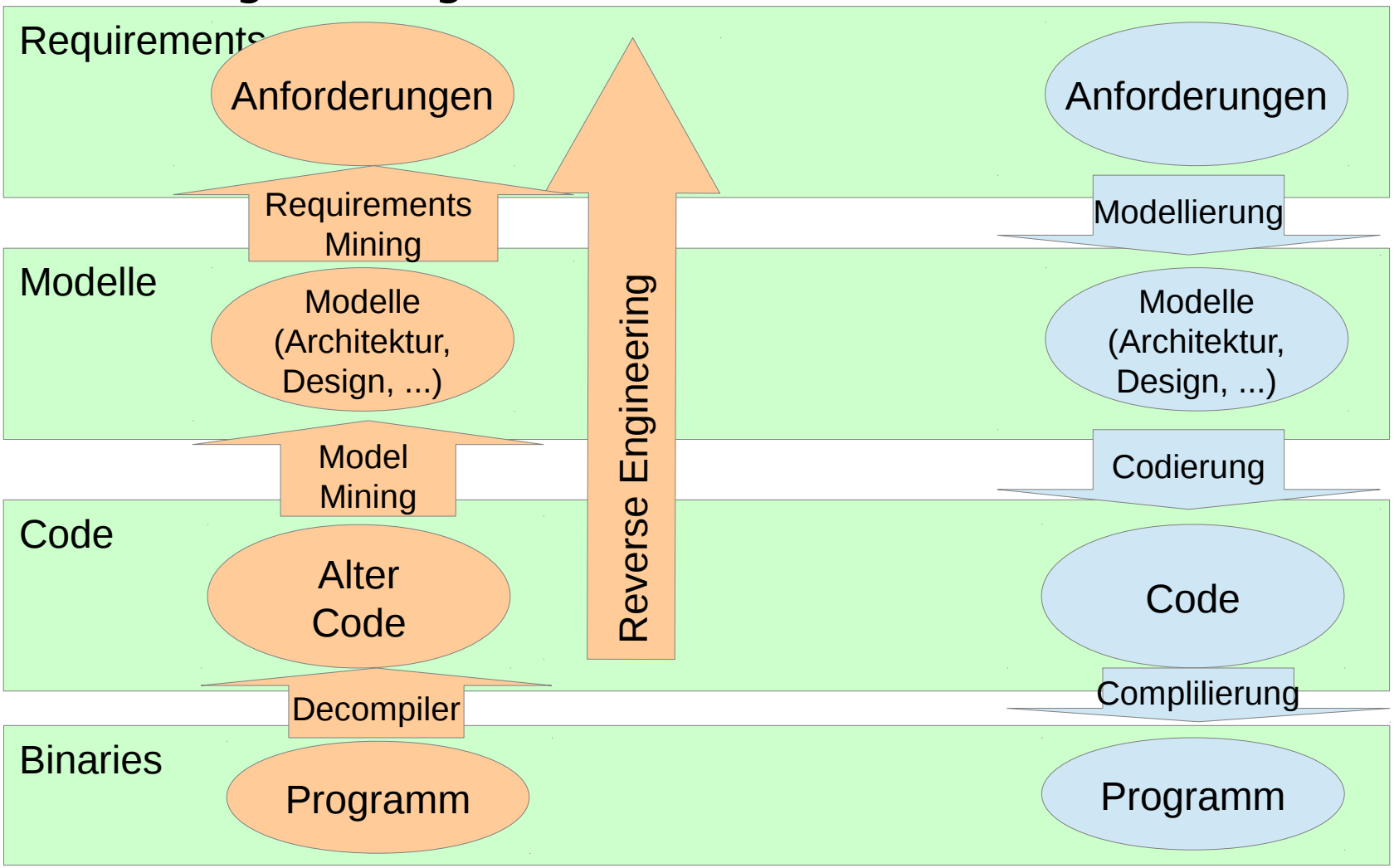
Das Reengineering-Problem



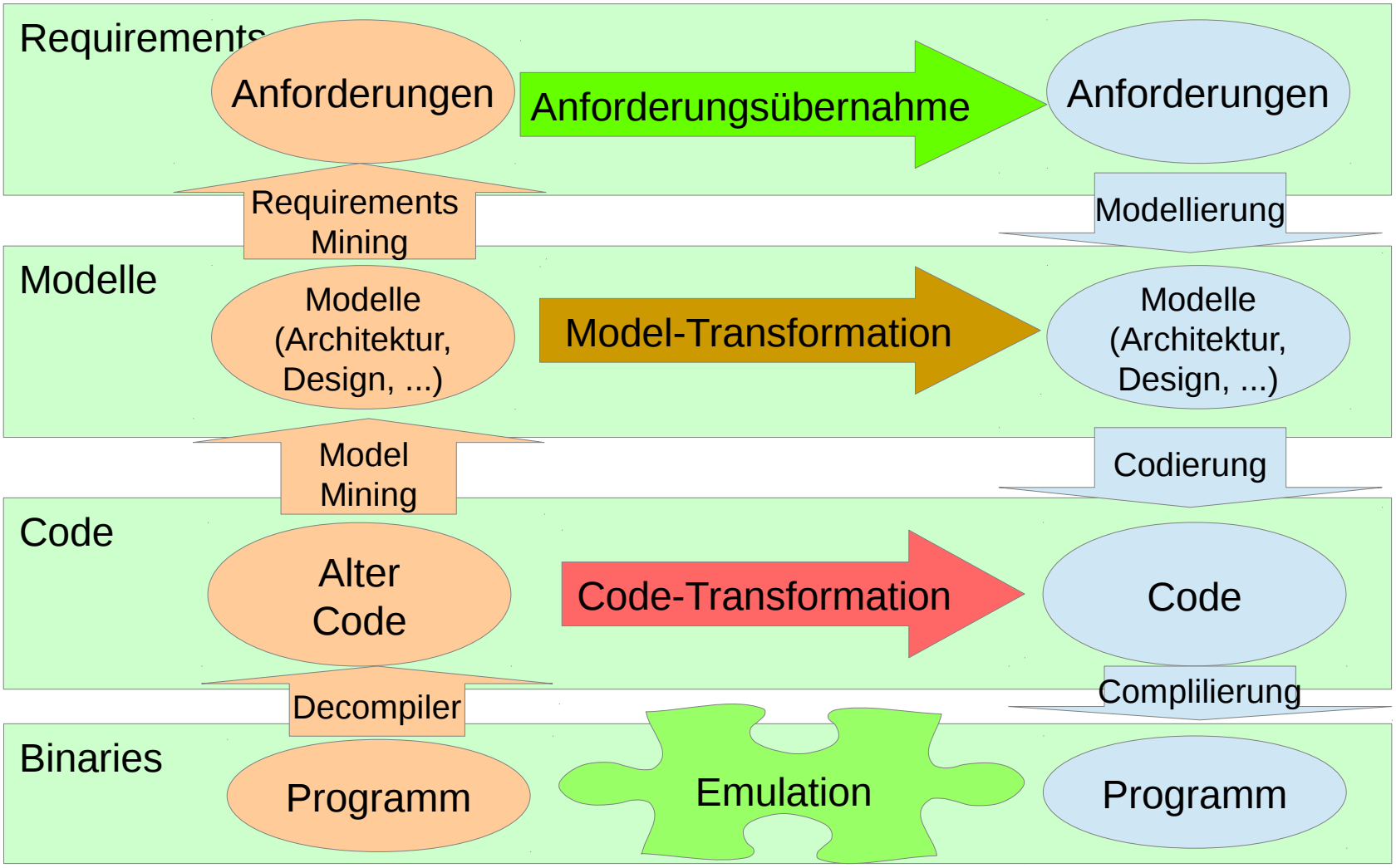
Das Reengineering-Problem



Das Reengineering-Problem

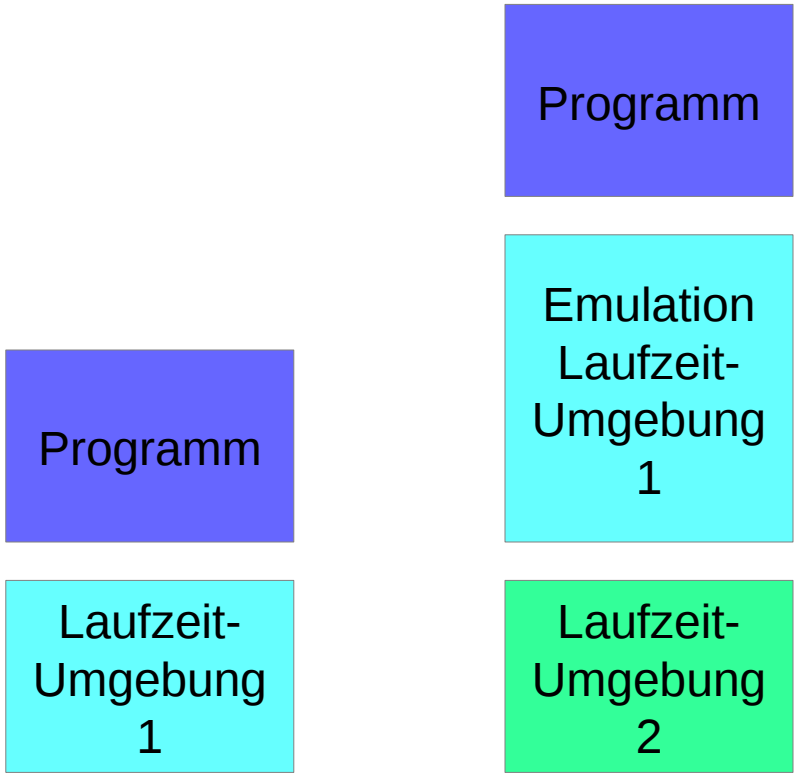


Transformationen „Aus Alt mach Neu“



Emulation

- Einfache Realisierung wenn Emulator bereitsteht
- Teilweise Performance Probleme
- Verschiedene Abstraktionsmöglichkeiten
  - Emulation von Bibliotheken (teilweise in Cygwin, Wine)
  - Emulation von Systembereichen (XP-Modus in Windows)
  - Emulation ganzer Rechner (Virtualisierung)



Decompiler

- Erzeugt „Code“ aus ausführbaren Dateien
- Probleme:
  - Keine lesbaren Namen
  - Nicht alle Strukturen können „rückübersetzt werden“
    - Optimierungen zerstören den linearen Ablauf
    - Beispiel zum Ausprobieren: Debuggen in Eclipse bei optimierenden Übersetzung

JD (Java Decompiler)

- Bei Java sind viele Informationen im Class-File enthalten
  - Klassennamen
  - Operationsnamen
- Wenn Debug-Informationen
  - Zeilennummern
  - Alle Variablen werden aufgelöst

Beispiel mit Debug-Infos

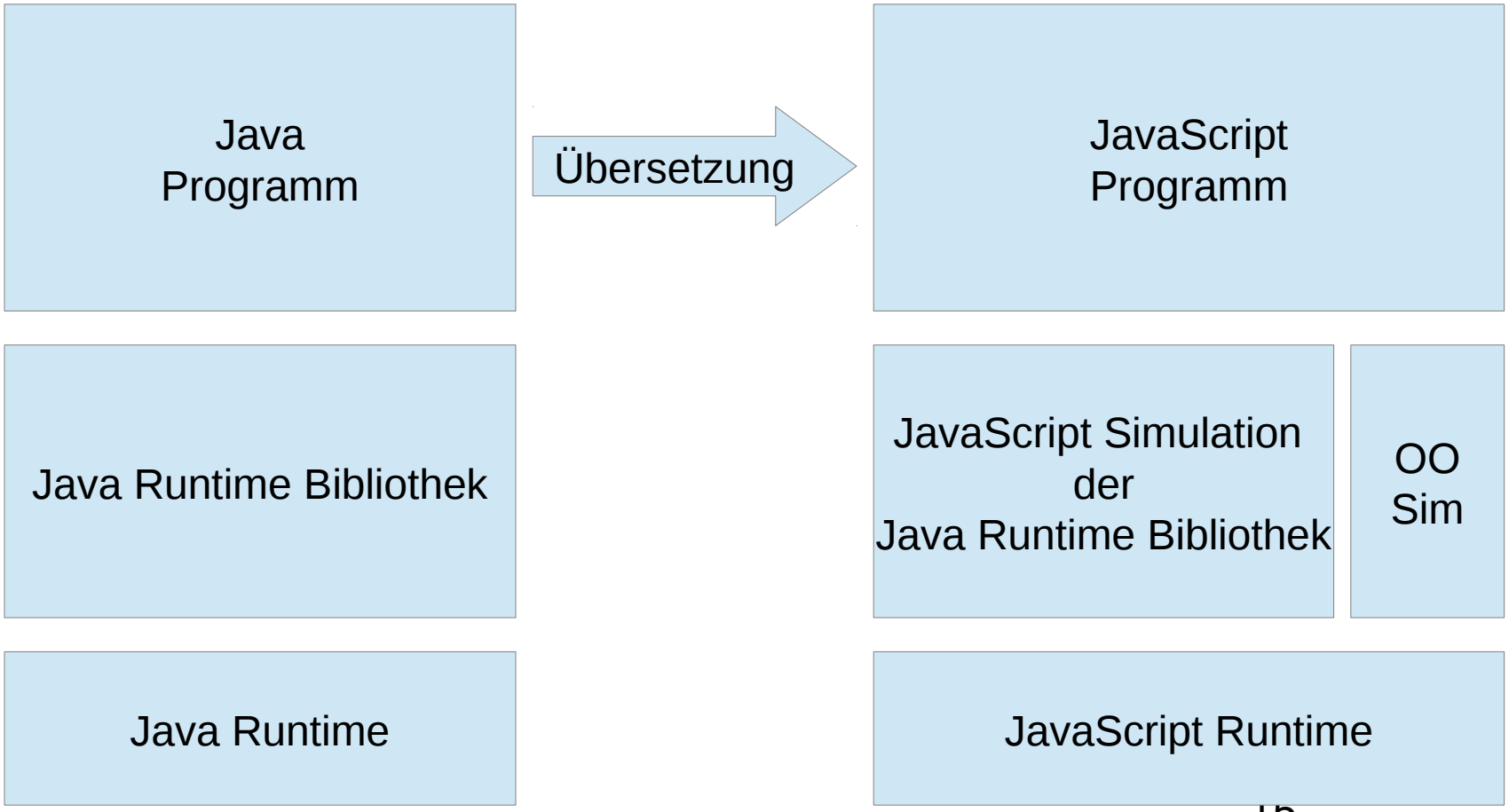
```
 void init()
{
    for (int i = this.maxValue; i >= 2; i--) {
    this.isPrim[i] = true;
    }
}

void findPrimNumbers()
{
     for (int i = 2; i * i <= this.maxValue; i++) {
     If (this.isPrim[i] != 0) {
          int j = i + i;
          while (j <= this.maxValue)
          {
           this.isPrim[j] = false;
           j += i;
          }
     }
     }
}
```

Code-Transformation

- Code wird in eine andere Sprache transformiert
- Herausforderungen:
  - Unterschiedliche Pradigmen
  - Unterschiedliche Bibliotheken
  - Unterschiedliche Semantik gleicher Syntax
- Beispiel: java2script – Java to JavaScript

java2script



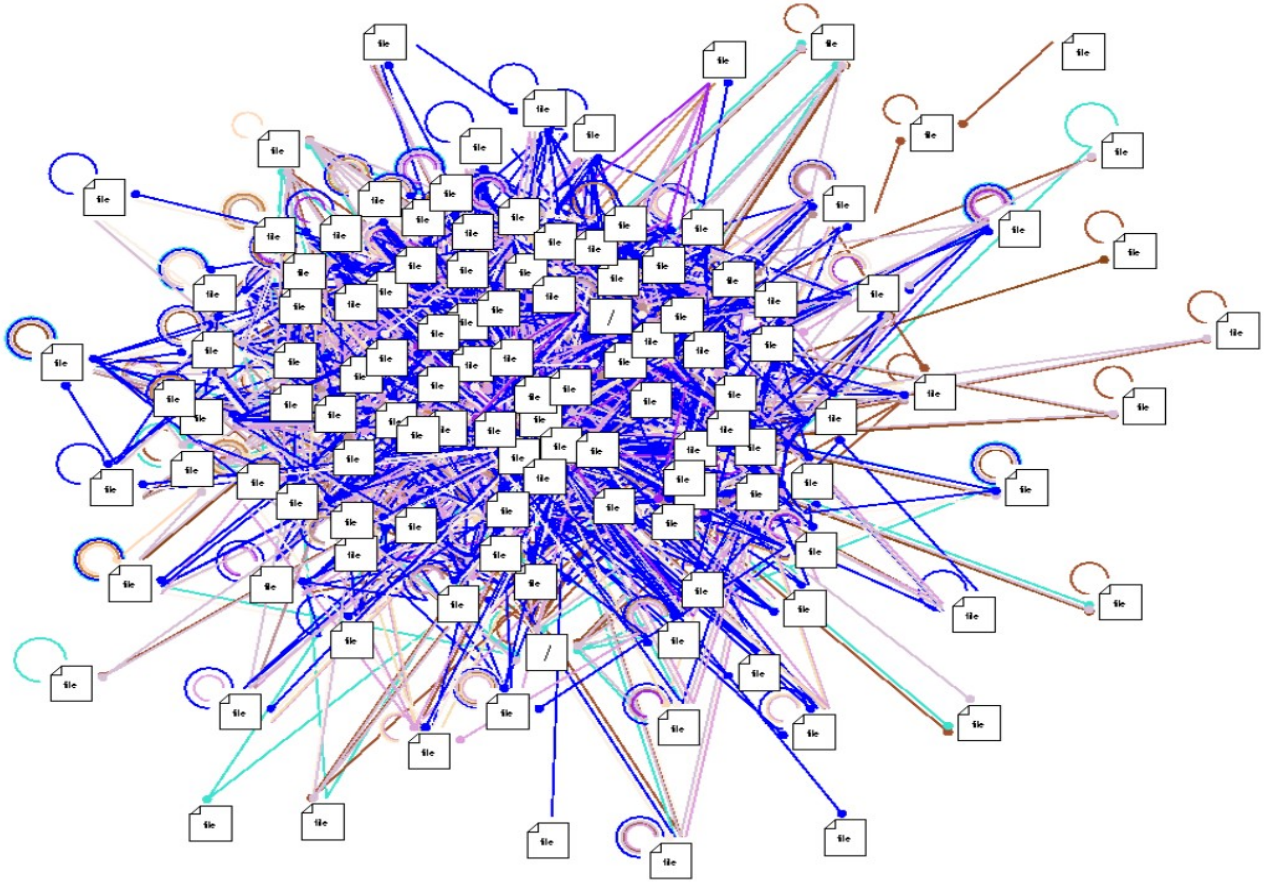
Einfaches Beispiel

- javascript:if(a="$wt.examples.controlexample.ControlExample@http://demo.java2script.org/org.eclipse.swt.examples/bin/",window["ClazzLoader"]!=null)w(a);else{var d=document,t="onreadystatechange",x=d.createElement("SCRIPT"),f=function(){var s=this.readyState;if(s==null||s=="loaded"||s=="complete"){w(a);}};x.src="http://archive.java2script.org/1.0.0-v20061220/j2slib.z.js";(typeof x[t]=="undefined")?x.onload=f:x[t]=f;d.getElementsByTagName("HEAD")[0].appendChild(x);void(0);}

Model-Mining - Code

- Erzeugen von Modellen aus dem Code
- Beispiele:
  - Erzeugen von UML-Diagrammen
  - Erzeugen von Prozessbeschreibungen
- Die Modelle sollten möglichst gut zum Programm passen
  - Java ↔ UML ok
  - C ↔ UML schwierig
  - Fortran77 ↔ UML schwierig
  - C ↔ Nasi-Schneiderman ok

Beispiel für automatische Umsetzung

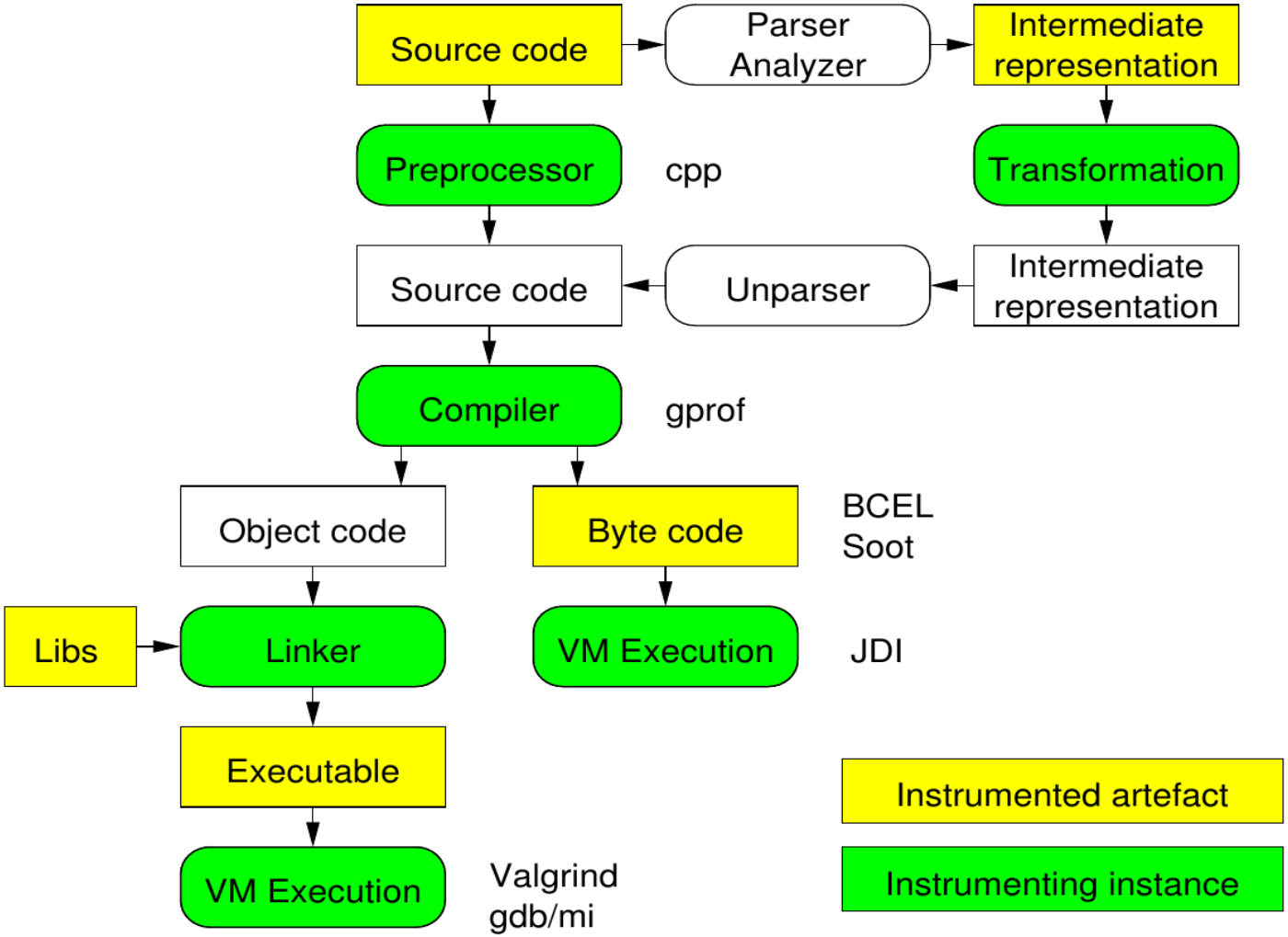


Model-Mining - Binaries

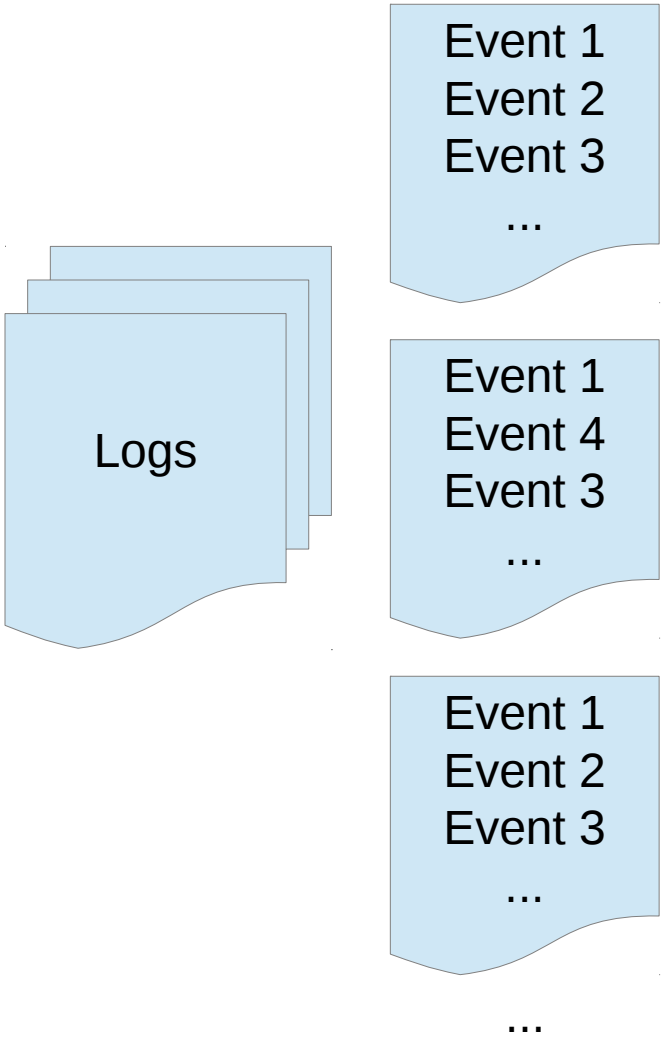
- 1. Möglichkeit:
  - Decompiler → Code → Model
- 2. Möglichkeit: Instrumentierung
  - Ergänzen des Programms um Analysepunkte
  - Beobachtung/Logging während des Laufes
  - Erstellen des Modells aus den Logging-Dateien

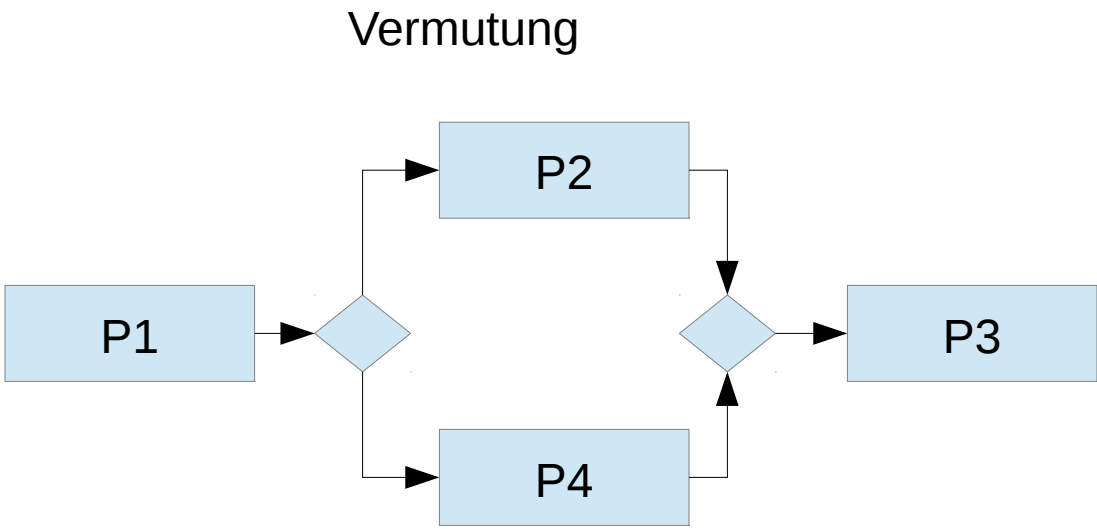
- Wird nicht nur zum Reengineering benötigt
  - Security-Analysen
    - Nachweis der Sourcen
    - Schädlingsanalyse
  - Performanceanalysen

Instrumentierung



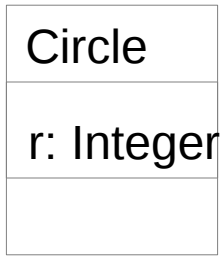
Process-Mining

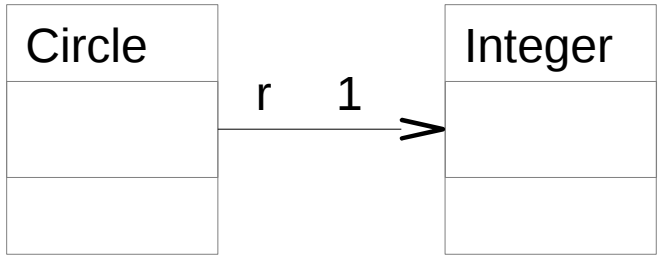




Java → UML-Klassendiagramm

- Übersetzung von Klassen einfach
- Was wird als Assoziationen, was wird als Fields dargestellt?
- Wie ist es mit den Multiplizitäten?
  - 1 meist noch eindeutig
- Wie mit anonymen Klassen umgehen?





Model-to-Model Transformation

- Ziel: Anpassung des Modells an die Zielsprache
- Teilweise Paradigmenwechsel
  - funktional → objektorientiert
  - deklarativ → imperative
- Teilweise Wechsel der Modellsprache
  - Jackson-Diagramme ↔ UML
  - BPMN ↔ UML

Requirements Mining

- Finden der Anforderungen die das Programm erfüllt
- Analyse der Funktion und des Nutzens
- Rekonstruieren der Einsatzszenarios

Anforderungsübernahme

- Meist direkte Übernahme der Anforderungen ins neue Projekt
- Häufig Anpassung
  - Entfernen von veralteten Anforderungen
  - Anpassen an neue Business-Prozesse
  - Optimieren

Nächste Woche:

Updates