technische universität
dortmund

Software-Engineering für langlebige Systeme

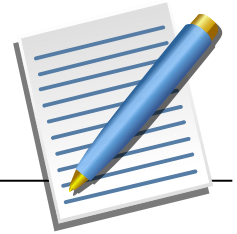
Thomas Ruhroth | Dortmund
SoSe 2014

VL1

- Softwareerosion
- Systemtypen

- Ziele:
  - Den Inhalt der Vorlesung kennen lernen.
  - Kennenlernen der grundlegenden Probleme durch Softwareerosion
  - Verstehen was langlebige Systeme sind.
  - Kennen lernen der Probleme von langlebigen Systemen.

Softwareerosion

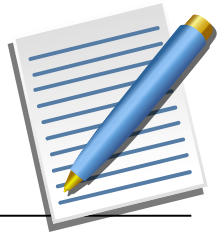


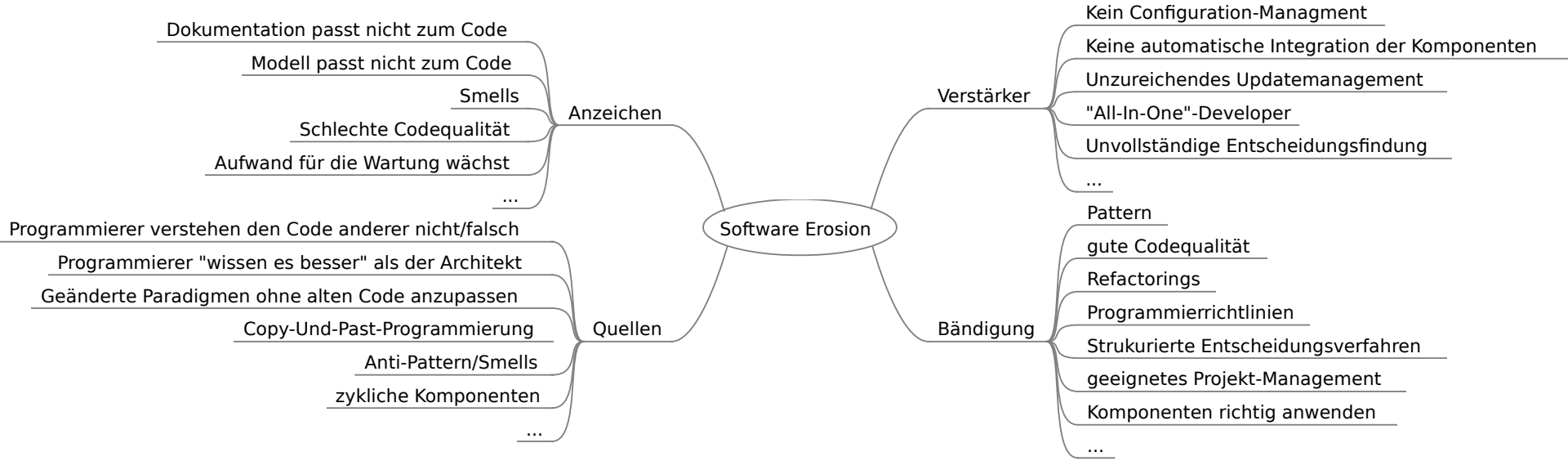
Softwareerosion

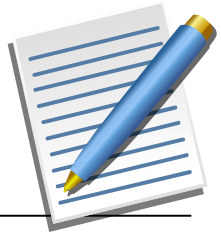
Softwareerosion ist der innere Strukturverlust von Software,

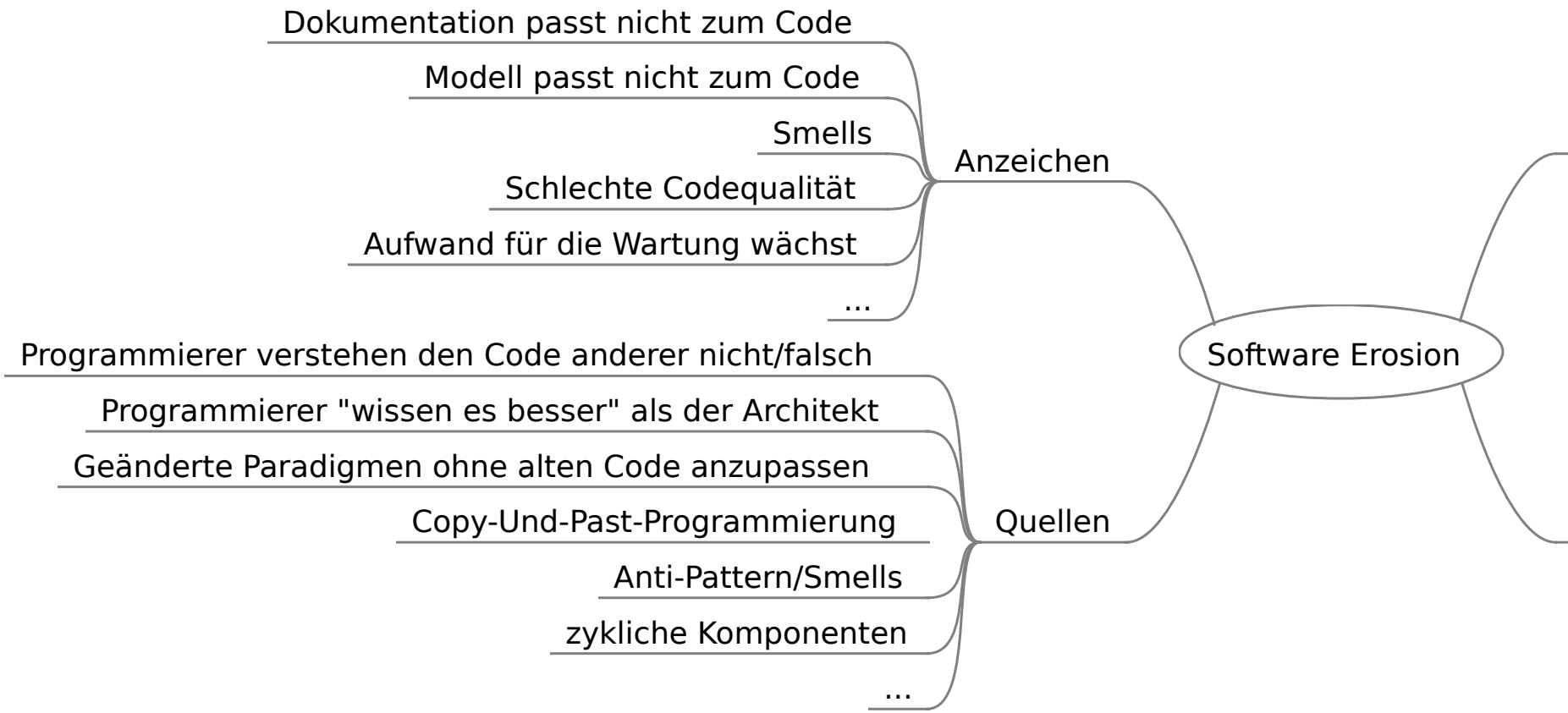
- der zu schlechter Wartbarkeit,
- schlechter Anpassbarkeit,
- schlechter Performance,
- Häufung von Fehlern und
- Häufung von Sicherheitsrisiken

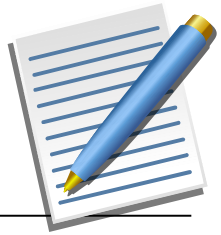
führt.

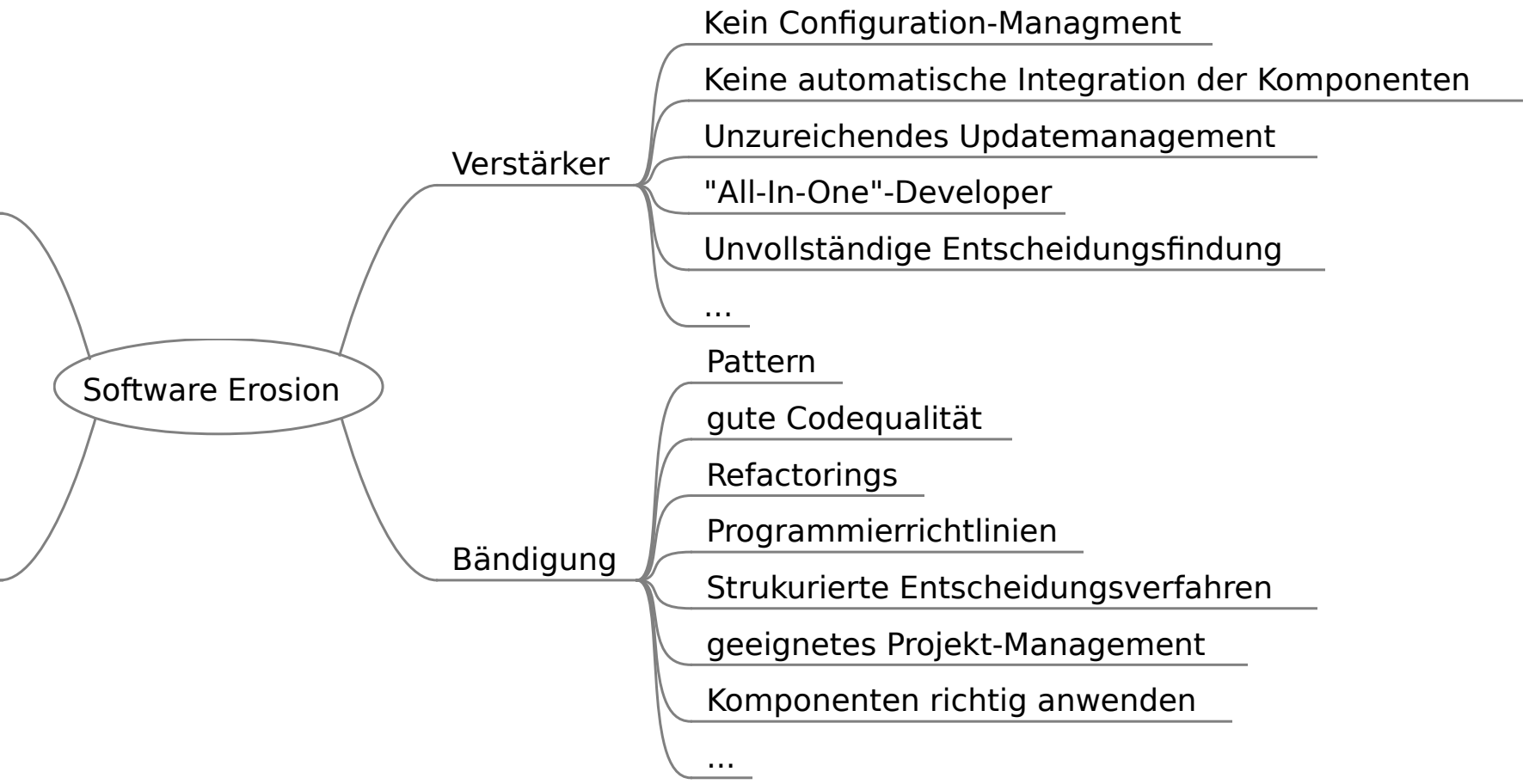



Software Erosion
Anzeichen
Dokumentation passt nicht zum Code
Modell passt nicht zum Code
Smells
Schlechte Codequalität
Aufwand für die Wartung wächst
...
Quellen
Programmierer verstehen den Code anderer nicht/falsch
Programmierer "wissen es besser" als der Architekt
Geänderte Paradigmen ohne alten Code anzupassen
Copy-Und-Past-Programmierung
Anti-Pattern/Smells
zykliche Komponenten
...
Verstärker
Kein Configuration-Managment
Keine automatische Integration der Komponenten
Unzureichendes Updatemanagement
"All-In-One"-Developer
Unvollständige Entscheidungsfindung
...
Bändigung
Pattern
gute Codequalität
Refactorings
Programmierrichtlinien
Strukurierte Entscheidungsverfahren
geeignetes Projekt-Management
Komponenten richtig anwenden
...




Dokumentation passt nicht zum Code
Modell passt nicht zum Code
Smells
Schlechte Codequalität
Aufwand für die Wartung wächst
...
Anzeichen
Programmierer verstehen den Code anderer nicht/falsch
Programmierer "wissen es besser" als der Architekt
Geänderte Paradigmen ohne alten Code anzupassen
Copy-Und-Past-Programmierung
Anti-Pattern/Smells
zykliche Komponenten
...
Quellen
Software Erosion




Software Erosion
Verstärker
Kein Configuration-Managment
Keine automatische Integration der Komponenten
Unzureichendes Updatemanagement
"All-In-One"-Developer
Unvollständige Entscheidungsfindung
...
Bändigung
Pattern
gute Codequalität
Refactorings
Programmierrichtlinien
Strukurierte Entscheidungsverfahren
geeignetes Projekt-Management
Komponenten richtig anwenden
...

Ein Beispiel

http://structure101.com/blog/2008/11/software-erosion-findbugs/

Problem im „findbugs“-Beispiel

- Komponentenstruktur geht verloren
- Problem aus dem Code nicht leicht ersichtlich
- Wartung wird immer Komplexer
  - Wenn eine Komponente geändert wird, müssen häufig auch andere Komponenten angepasst werden.
  - Wiederverwertung schwierig
  - Fehlereingrenzung schwierig

Wie feststellen?

- Metriken
  - Lack of Cohesion of Methods
  - Dependency Structure Matrix
  - ...
- Strukturgraphen (wie im Beispiel)
- Direkte Code-Analyse

Wie beheben und vermeiden?

- Refactoring
- Pattern
- Evolutionsanalysen
- Co-Evolution
- Code-Guidlines
- Systemabstraktionsschichten
- Verbinden von Systemen
- Komponentendesign

Plan

Meta-Ebene

Projekt-Management, Arbeitstechniken

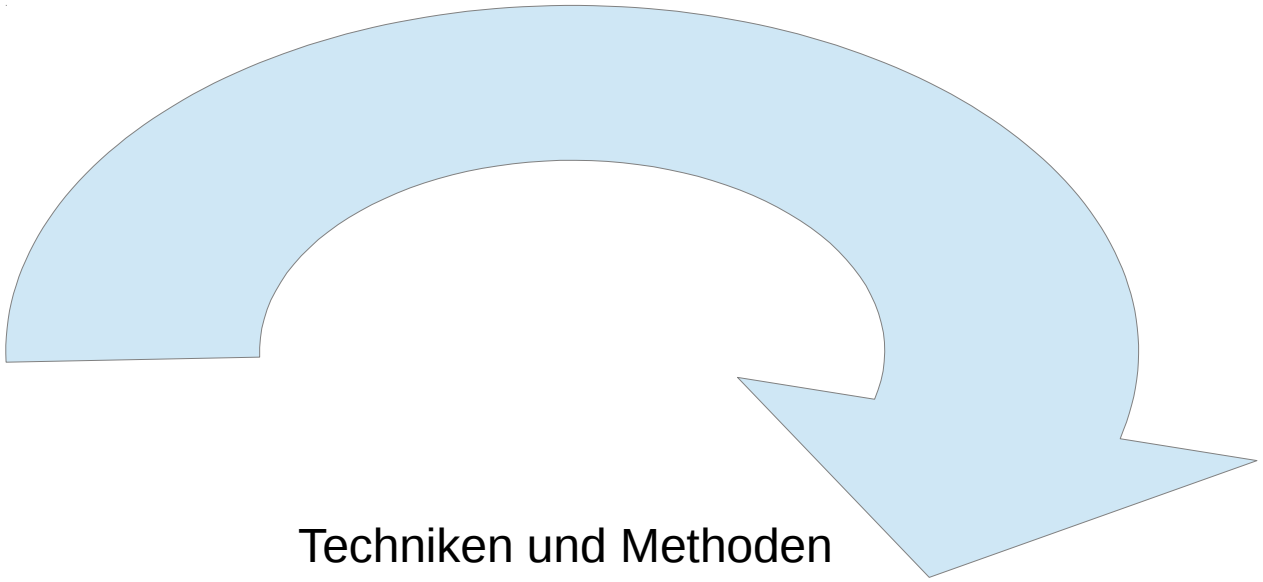
MDD-Ebene

Requirments

Design

Code

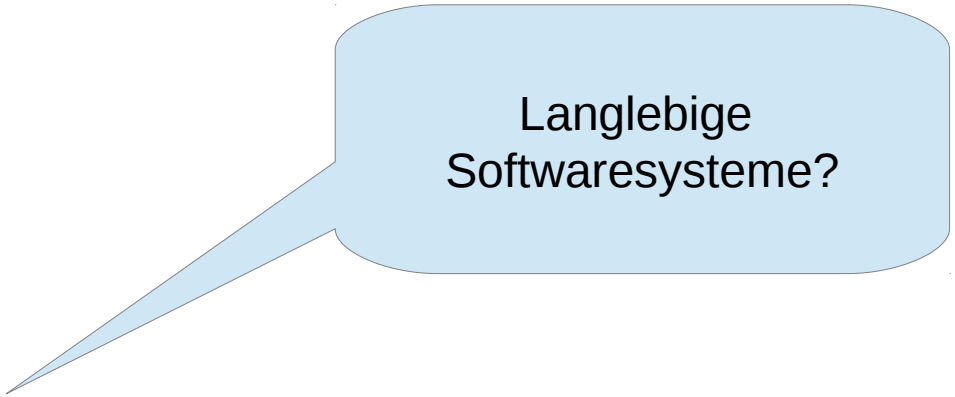
Test



Techniken und Methoden
zur Verlangsamung von Softwareerosion
und zur Softwarerestauration

Semester-Verlauf

Softwaresysteme



- Betriebssysteme
- Eingebettete Systeme
- Datenbanksysteme
- Informationssysteme
- ....

- Definition/Abgrenzung
- Designeinflüsse/Anforderungen

Betriebssysteme

- Ein Betriebssystem ist eine Sammlung von Computerprogrammen, die die Systemressourcen eines Computers wie Arbeitsspeicher, Festplatten, Ein- und Ausgabegeräte verwaltet und diese Anwendungsprogrammen zur Verfügung stellt. Das Betriebssystem bildet dadurch die Schnittstelle zwischen den Hardwarekomponenten und der Anwendungssoftware des Benutzers.

Andrew S. Tanenbaum: Moderne Betriebssysteme. Pearson Studium, 3., aktualisierte Auflage, ISBN 978-3-8273-7342-7

Betriebssysteme - Designeinflüsse

- Hardware Abstraktion
  - Anwendungen möglichst unabhängig von der Hardware sein.
- Sicherheit
  - Isolation von Daten, Prozessen und Nutzern
  - Authentisierung, Rechtemanagement
- Stabil und Robust
- Ressourcenverwaltung und Konfliktauflösung
  - Drucker, Festplatten etc.

Eingebettete Systeme

- Definition 1:
  Ein Rechensystem, das eine Funktion oder einen Funktionsbereich überwacht: wissenschaftliche, technische und/oder industrielle Steuerung
- Definition 2:

  Jedes in einem Produkt versteckte Rechensystem, das selbst kein Rechner ist : Konsumgüter, Haushaltegräte, Fahr- und Flugzeuge, . . . , Waffen
- Definition 3:
  Ein zur Durchführung einer spezifischen Aufgabe entwickeltes Rechensystem, wenn auch mit Auswahl und Optionen

Datenbanken

- Geschwindigkeit
- Transaktionssicherheit
- Datensicherung
- Abfrageoptimierung

Eingebettete Systeme - Designeinflüsse

- Integration
  - Mikrocontroller
  - Peripheriebausteine, ...
- Begrenzte Ressourcen
  - Wenig Speicher
  - Wenige Anschlüsse, ...
- Stromverbrauch
- Echtzeitanforderungen
  - Hohe Verfügbarkeit
  - Definierte Antwortzeiten
- Betriebssicherheit
- Stückpreis
- Entwicklungsumgebung
  - Entwicklungsarchitektur ungleich Laufzeitarchitektur

Langlebige Softwaresysteme

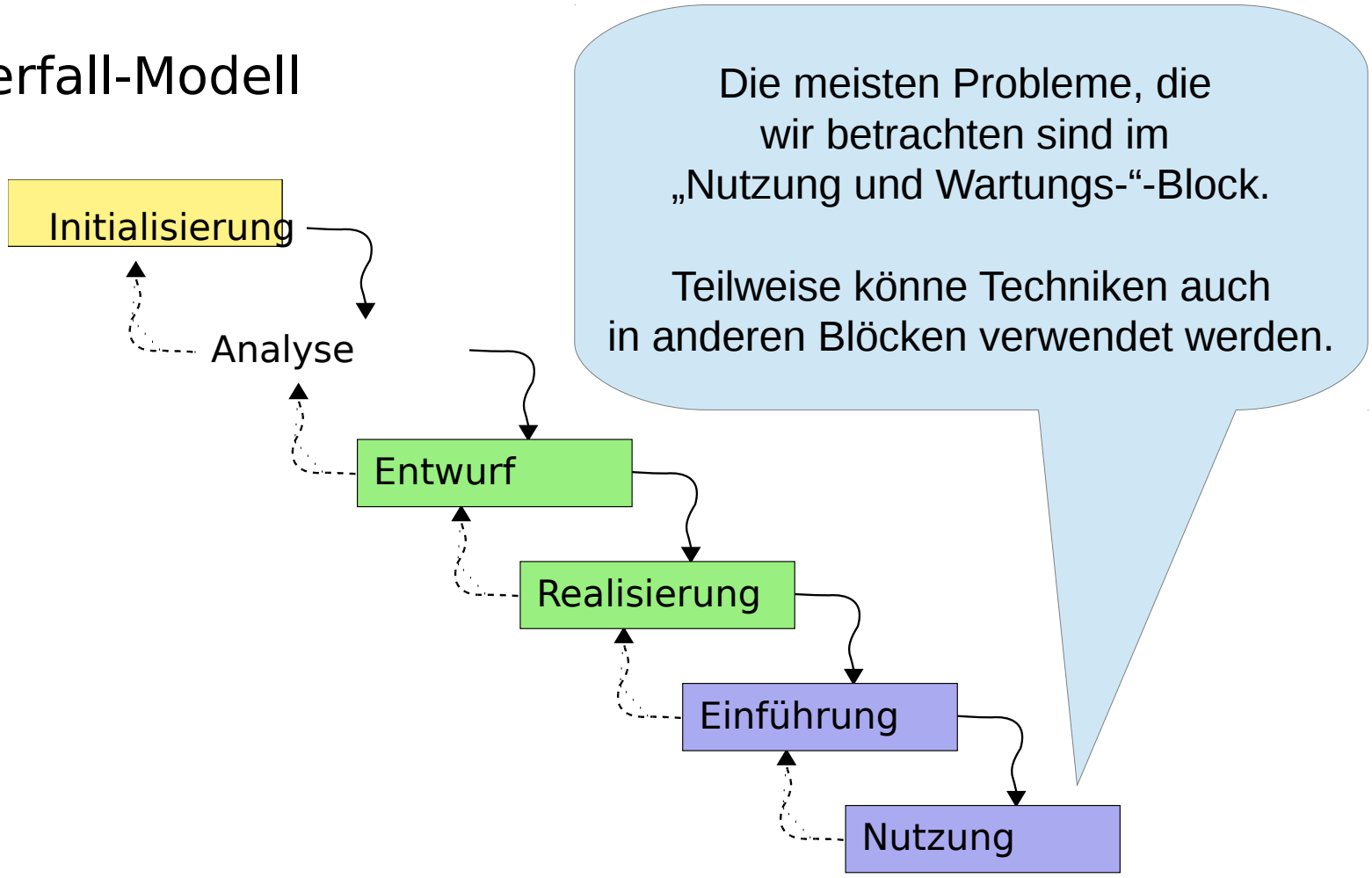
- Langlebigkeit ist eine Eigenschaft in verschiedenen Systemklassen
  - Betriebssysteme
  - Steuerungssoftware
- Meist nicht langlebige Systeme
  - Simulations-Prototypen

Langlebige Softwaresysteme

- Langlebige Softwaresysteme sind Softwaresysteme, die solange eingesetzt werden, dass diese regelmäßig und über einen langen Zeitraum an sich ändernden
  - Hardware,
  - Softwareumgebungen (einschl. BS und Libs) und
  - Benutzungsanforderungen

  angepasst werden müssen und über diesen Zeitraum gefundene Fehler gefixt werden müsse.

Wasserfall-Modell

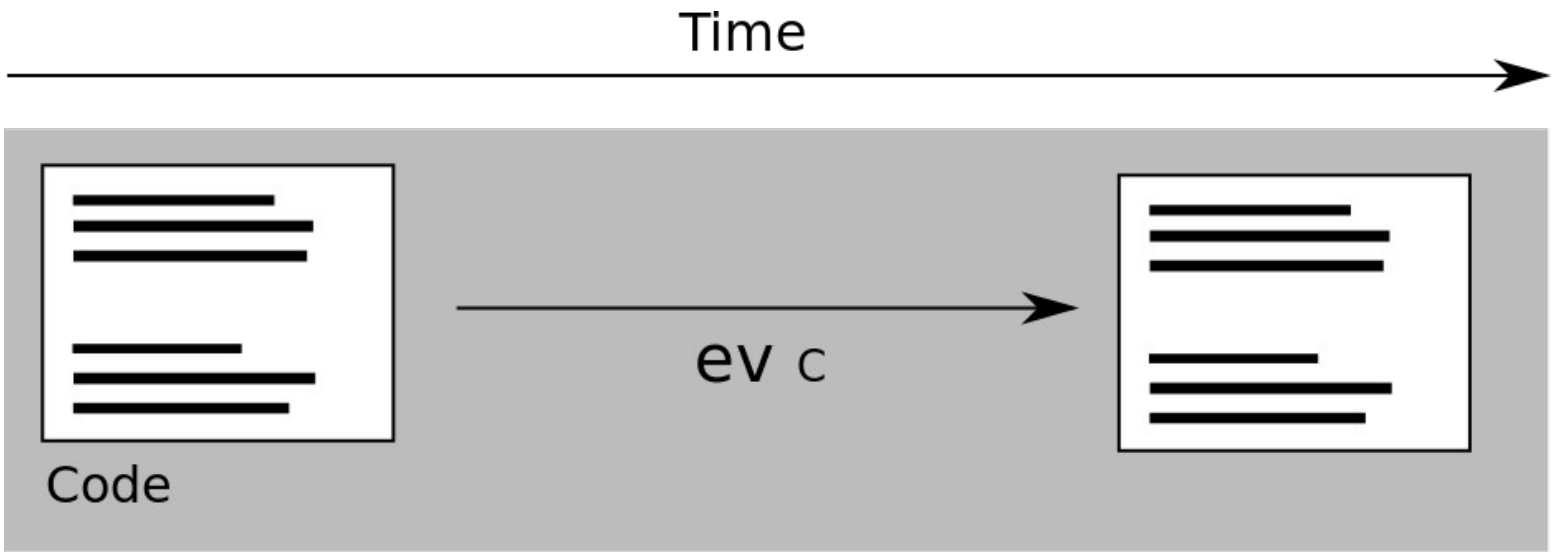


Nutzungs- und Wartungsphase - Softwareänderungen

- Bugfix
  - Beseitigt **nicht-kritische** Fehler
  - häufig kleine Änderung
  - **zügige** Auslieferung
- Hotfix
  - Beseitigt **kritische** Fehler
  - **Sofortige** Auslieferung
- Update
  - Hinzufügen und Ändern von Features/ Systemumgebungsunterstützung
  - Häufig definierte Zyklen, meist mittel- bis langfristig
- Relaunch
  - Größere Umbauten/ Featureänderungen/ Systemumgebungsänderungen

Softwareevolution

Evolution

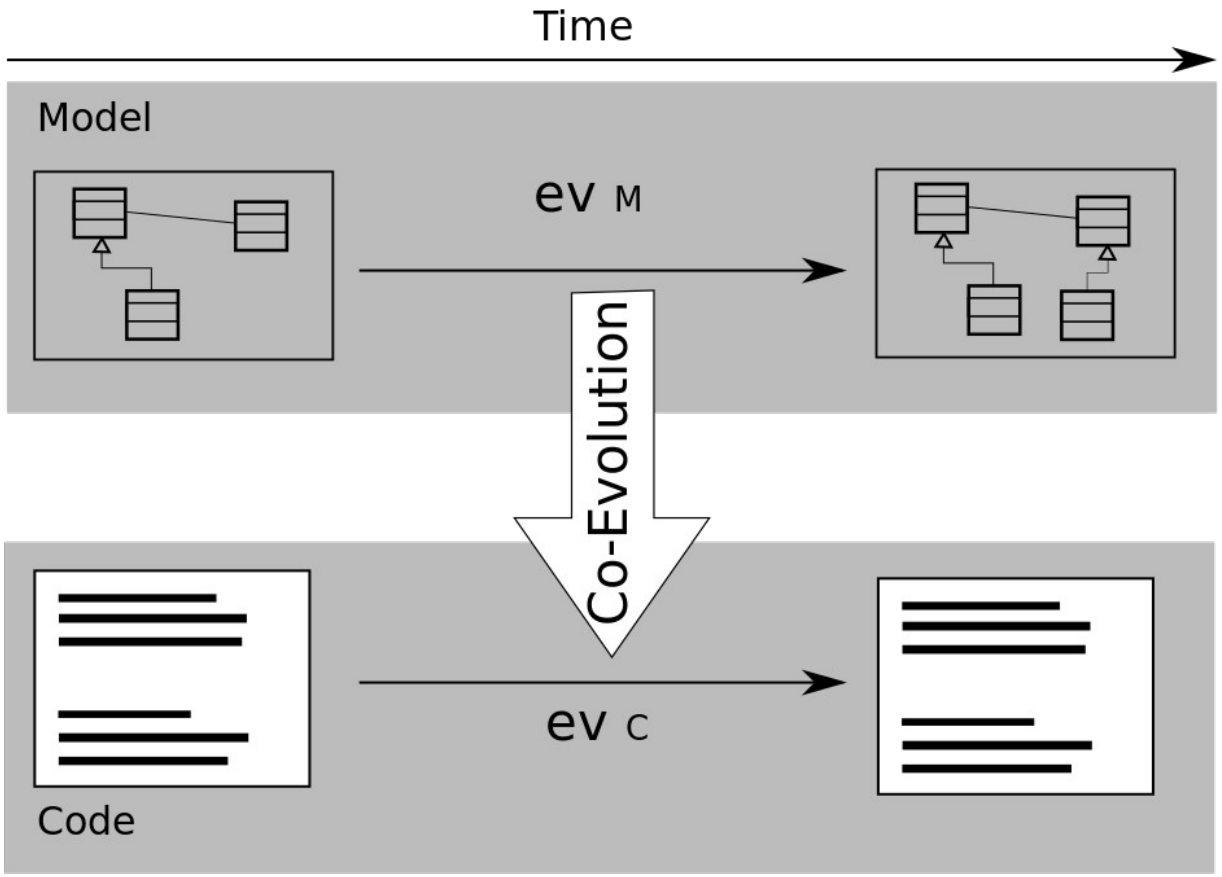


Evolution

- Evolution beschreibt die schrittweise Veränderung von Software.

- Nach einer Evolution
  - Gültiger Code/gültige Modelle
    - Syntax wird eingehalten
    - Wohlgeformtheit ist sichergestellt
- Aneinanderreihung von Evolutionen ergibt wieder eine Evolution
- Verhaltenserhaltende Umformungen einschl. Refactorings sind Evolutionen.
- Achtung: Der Begriff Evolution ist überladen!
  - Evolution als Beschreibung der Änderung im Ganzen
  - Evolution als Änderungsfunktion

Achtung bei Modellen



Co-Evolution

- Wie müssen zwei Artefakte gleichzeitig einer Evolution unterworfen werden, damit diese noch Konsistent zueinander sind.
- Sehr aktives Forschungsthema
- Viele Real-World-Probleme aus der Industrie

Warum reicht nicht ein einfacher Katalog von Regeln?

Welchen Programmierstiel ist besser?

```
public static int iterFibo(int i) {
int a = 1, b = 1, tmp = 1;
for (int j = 0; j < i; j++) {
b = tmp;
tmp = a +b;
a = b;
}
return b;
}
```

```
public static int recurFibo(int i) {
if(i <= 1)
return 1;
return recurFibo(i-1) + recurFibo(i-2);
}
```

Performance vs. Verständlichkeit

Rekursiv

- Performance
  - Teure Funktionsaufrufe
  - Mehrfachberechnung von Werten
- Verständlichkeit
  - Algorithmus schnell ersichtlich

Iterativ

- Performance
  - Meist gut
- Verständlichkeit
  - Zusätzliche Hilfskonstrukte
    - Mehr Variablen
    - Steuerkonstrukte

Es kommt drauf an!

- Technisch
  - Programmiersprache
  - Programmierparadigma
  - Systemarchitektur
- Programmierpsychologisch
  - Verständlichkeit
  - Modelleierbarkit
  - Wartbarkeit

Rekursion-Probleme

- Stacküberlauf schwierig zu handhaben
  - Microsoft verbietet die Nutzung von Rekursion in bestimmten kritischen Bereichen z.B. Kernel
- Rekursion nicht immer verfügbar
  - Abhängig von Hardware-Umsetzung

Softwaremanagement für langlebige Systeme
(angelehnt an ITILv3)

ITILv3

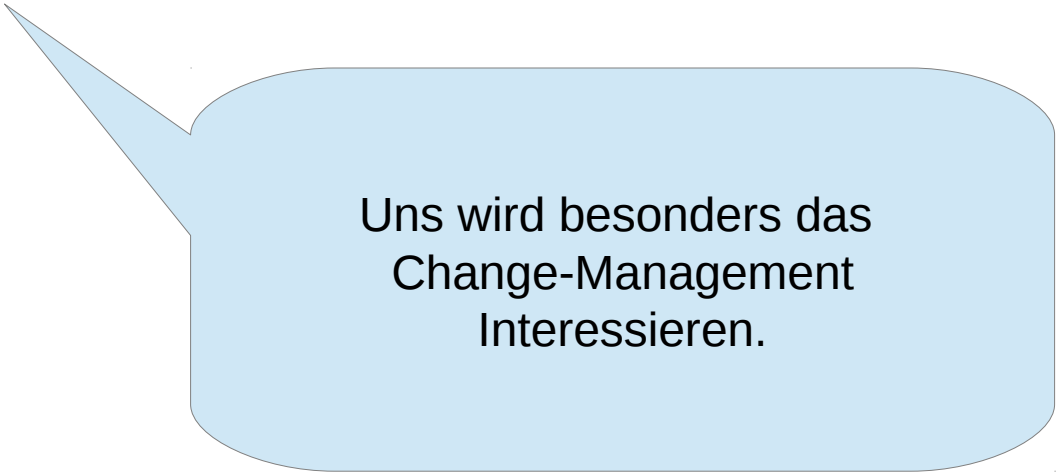
- IT-Service-Management
- De-facto-Standard
  - Wird in vielen Stellenausschreibungen gefordert

Ziele von ITIL

Verbesserung

- der Effizienz,
- der Qualität und
- der Wirtschaftlichkeit

der jeweiligen IT-Organisation.



ITTv3 - Themen

- Change/ Request-Management
  - Wer darf Changes anfordern?
  - Wer entscheidet bei der Umsetzung?
  - Wie werden Changes ausgerollt?
- Supporttypen
  - Pilot-Support
  - Early-Life-Support
  - Standard-Support
  - Legacy-Support

Fragen zur Inhaltsübersicht/Einleitung?