

Refactoring von Legacy Code

Von Alexander Dahmen und Tom Niklas Brell

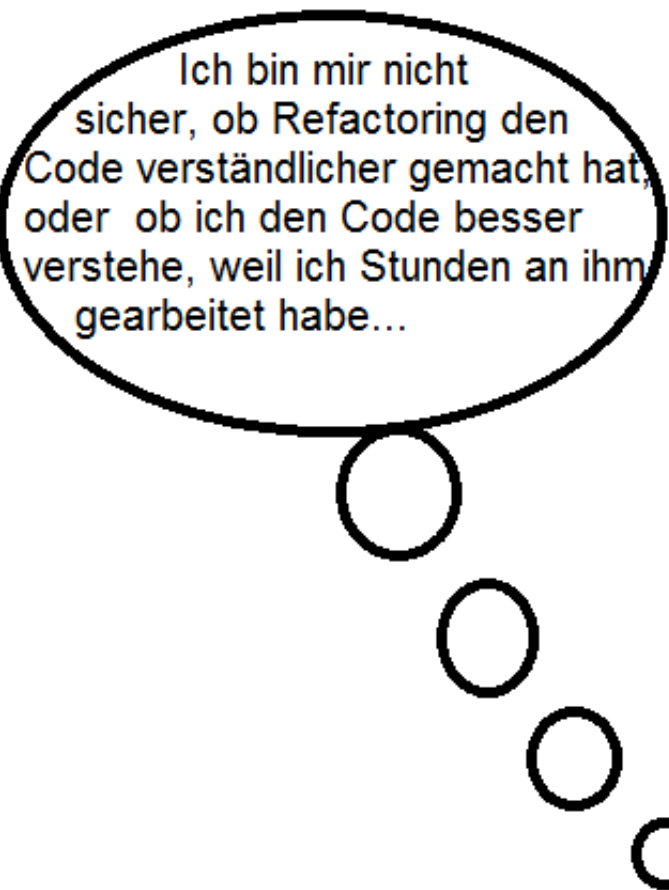
Wissenschaftliches Arbeiten in der Informatik 1

Prof. Dr. Inge Schestag

Sommersemester 2016

Gliederung

- Grundlagen
 - Definition und Merkmale
 - Gründe für das Arbeiten mit Legacy Code
- Einführung in das Refactoring
 - Unit Tests
 - Dependency Breaking
 - Arbeiten mit altem Code
- Automatisierte Tools
 - IDE-interne Tools
 - Externe Tools



Ich bin mir nicht sicher, ob Refactoring den Code verständlicher gemacht hat, oder ob ich den Code besser verstehe, weil ich Stunden an ihm gearbeitet habe...

Merkmale und Definition

- Veralteter, schwer verständlicher Code
- Keine oder wenige Tests
- Alte oder nicht weit verbreitete Sprache
- Schlechte Struktur, toter Code
- Unter Umständen sehr wertvoll

Warum arbeiten wir mit Legacy Code?

- Wenn der Code verstanden ist, kann er sehr wertvoll sein.
- Oft effektiver und weniger zeitaufwendig als komplett neuen Code zu schreiben.
- Bessere Struktur und Hinzufügen neuer Features auch durch Refactoring möglich.

Refactoring

- Verbesserung von Code, ohne Veränderung nach außen
- Gründe
 - Struktur verbessern
 - Feature hinzufügen
 - Effizienz steigern
- Voraussetzung für das Arbeiten mit altem Code: Tests

Unit-Tests (1/2)

- Automatisierte Tests einzelner Teile heißen Unit Tests.
- Viele Frameworks vorhanden: JUnit, CppUnit, etc.
- Qualitäten: Unit Tests ...
 - laufen schnell (0.1 Sekunden ist ein langsamer Test)
 - helfen uns Probleme zu lokalisieren

Unit-Tests (2/2)

- Problem: Komponenten funktionieren isoliert nicht immer → Abhängigkeiten
- Verantwortungen trennen nennt man „Dependency Breaking“
- Nach Isolation der Komponenten: Tests schreiben, dann Refactoring
- Test-Driven Development: Für neue Funktionen erst Tests, dann Funktionalität schreiben

Dependency Breaking (1/2)

- Mögliche Abhängigkeiten:
 - Lange Kette von Objekten, die wiederum Objekte brauchen
 - Versteckte Zugriffe auf andere Teile des Systems
 - Verwendung externer Ressourcen wie Datenbanken oder Netzwerke

Dependency Breaking (2/2)

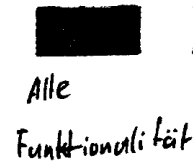
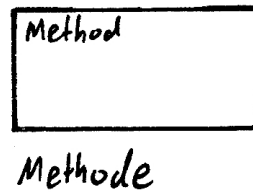
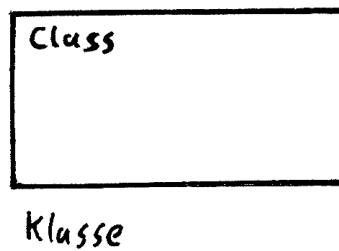
- Techniken zum Trennen:
 - Interface extrahieren (→ Grafik von UML Diagramm)
 - Bei Instantiierung innerhalb von Konstruktor: Parameter zum Konstruktor hinzufügen
 - Methode enthält problematische Zugriffe: Aufruf extrahieren und überschreiben

Arbeiten mit altem Code

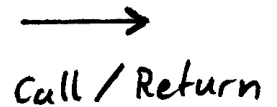
- Verstehen von Code:
 - Skizzen/Notizen anfertigen
 - Code ausdrucken und Gedanken hinein zeichnen
 - Code kopieren und in Kopie arbeiten, diese danach verwerfen

Hinzufügen von Features (1/4)

Legende:



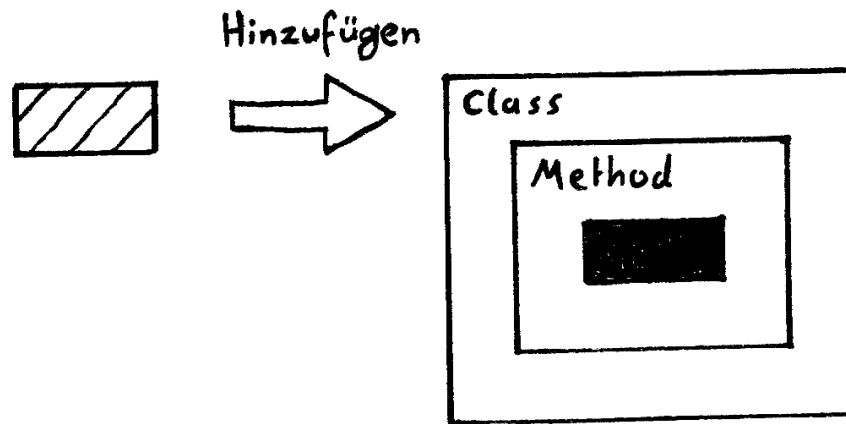
NEW
Neu
hinzugekommen



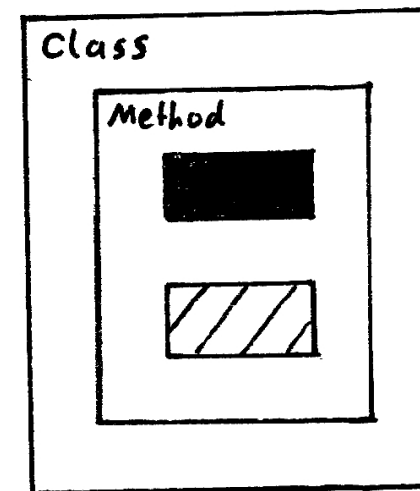
Legende

Hinzufügen von Features (2/4)

Problem :



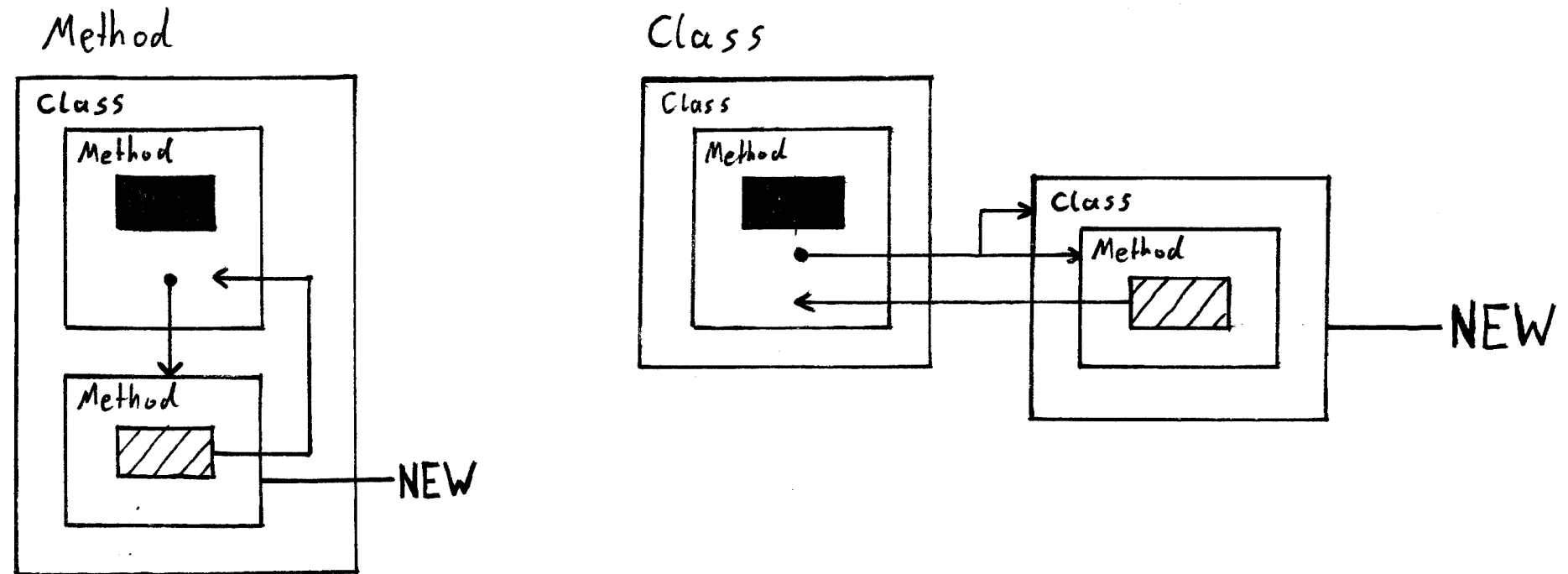
Schlechte Vorgehensweise :



Problem und schlechte Vorgehensweise

Hinzufügen von Features (3/4)

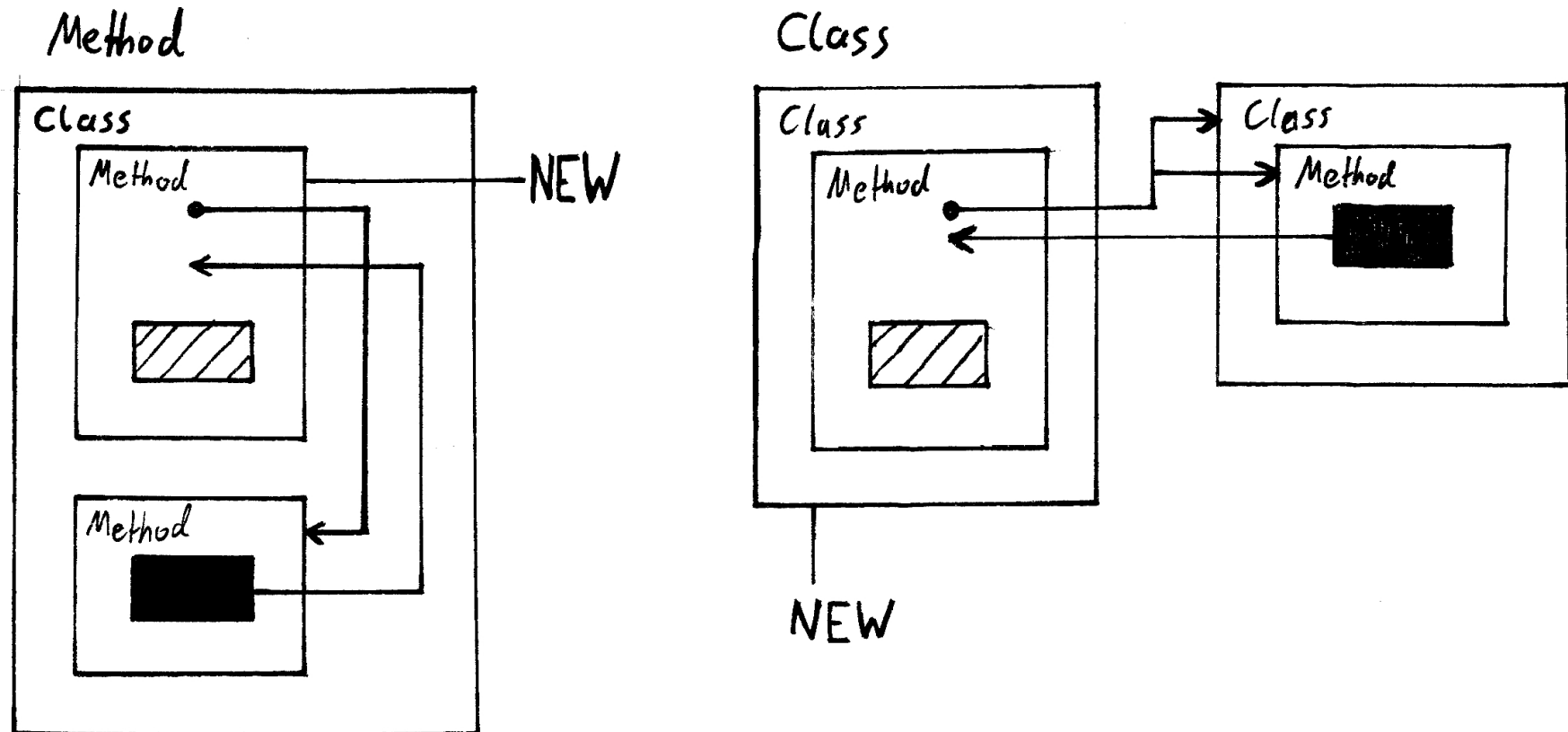
SPROUT :



Sprout-Methode und Sprout-Klasse

Hinzufügen von Features (4/4)

WRAP :



Wrap-Methode und Wrap-Klasse

IDE-interne Tools

- Editoren wie Eclipse verfügen über eine große Anzahl an Möglichkeiten für das *Refactoring*...
- ...von sehr einfachen Tools wie *Rename*, bis hin zu komplexeren wie *Extract Method*.
- Es gibt z.B. Tools für das Verschieben von Klassen/Methoden oder für das automatische Erzeugen von Getter-/Setter-Methoden.

Beispiel: *Inline*

Fügt den Wert einer Variable anstatt des Namens ein.

```
int wert1=2+2;
```

```
int wert2= wert1 +3;
```

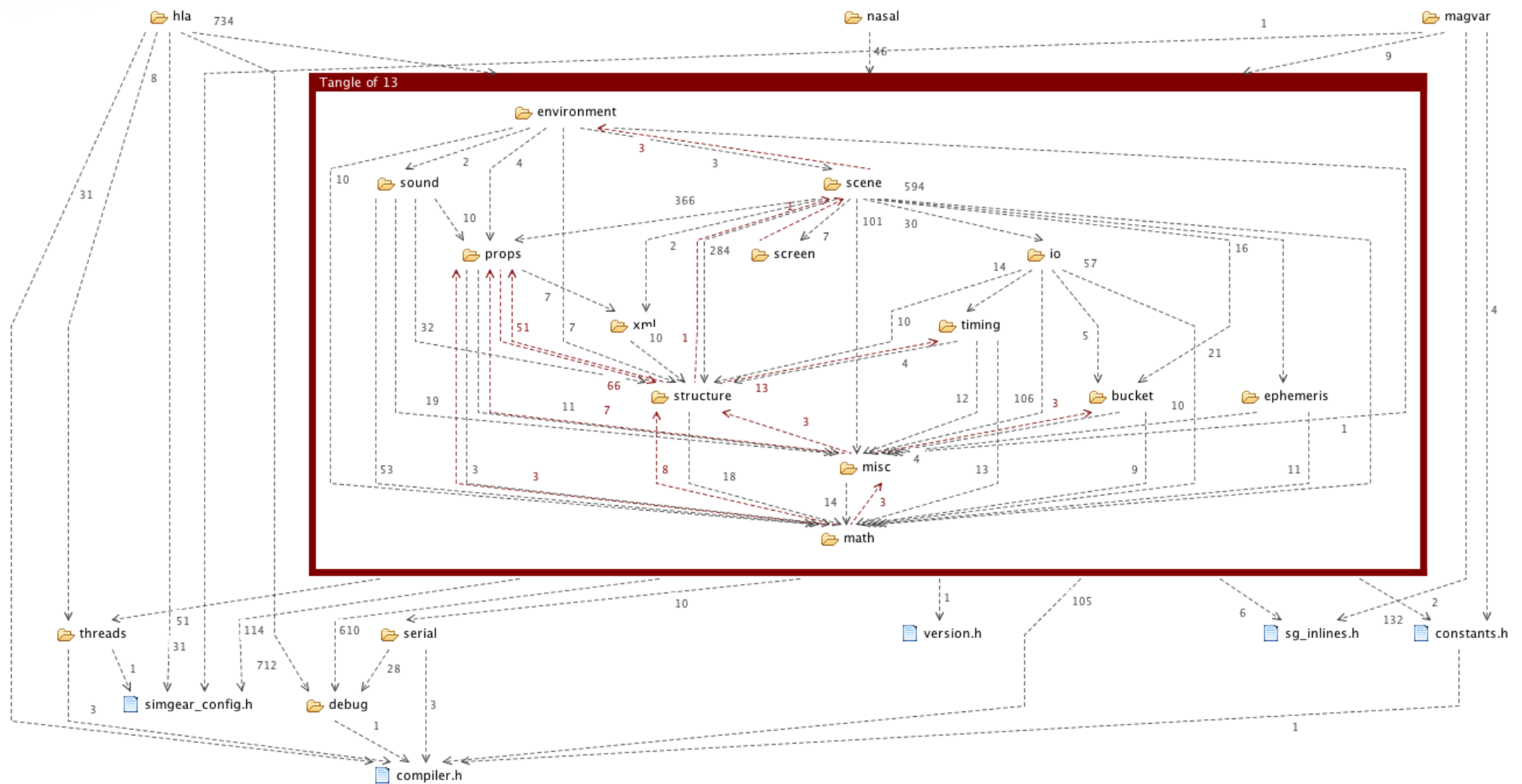
```
wird zu => int wert 2 = 2+2+3;
```


Externe Tools - *PMD*

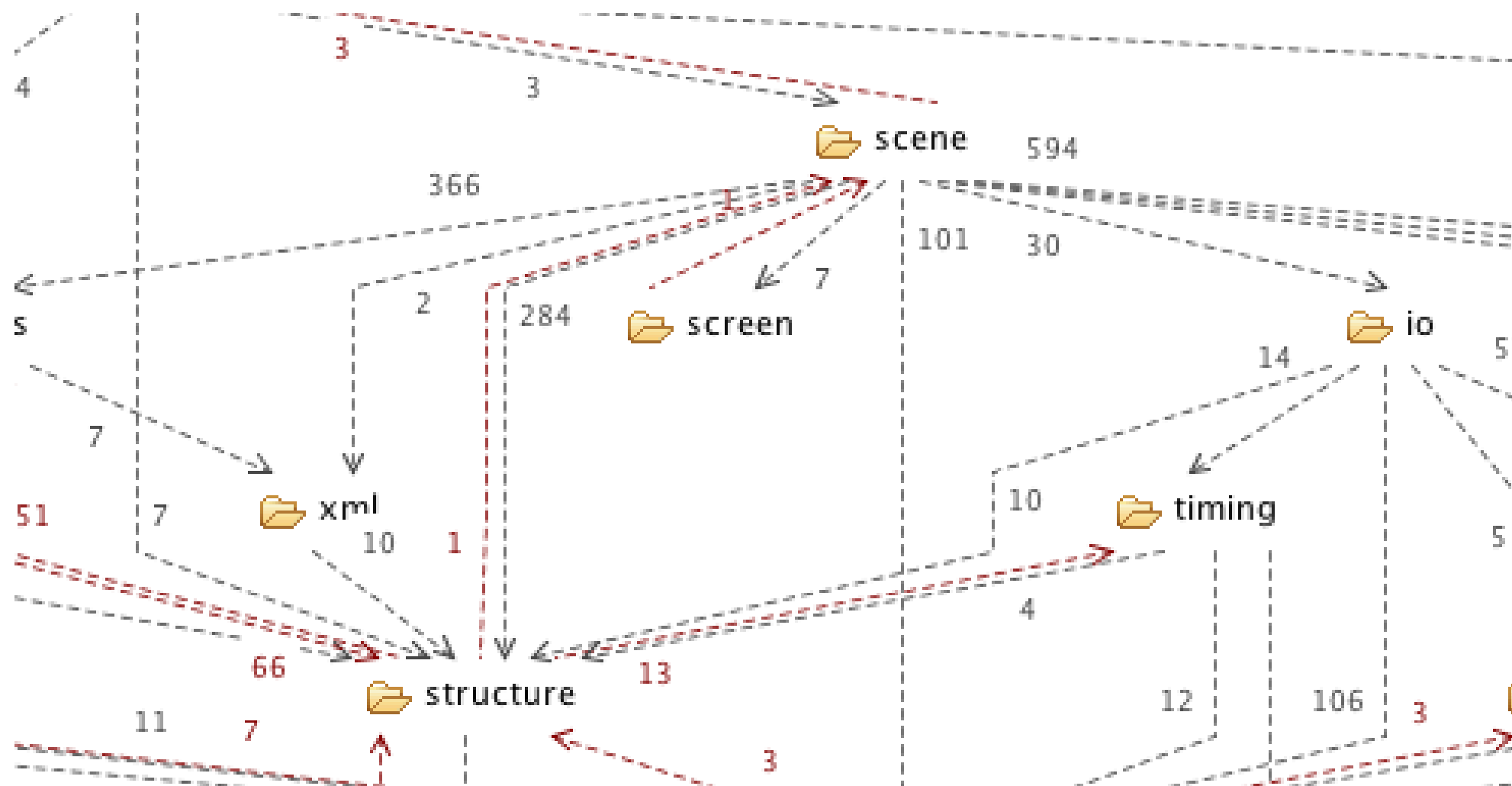
- Sucht keine echten Fehler, sondern hilft ineffizienten Code zu verbessern
- Sucht z.B. leere try/catch/switch-Blöcke, die Bugs verursachen können
- Es ist möglich, den Satz an Regeln zu erweitern und an die eigenen Bedürfnisse anzupassen.

Externe Tools – *Structure 101*

- Analysiert das Projekt im Gesamtbild
- Abhängigkeiten und die Komplexität werden in Grafiken veranschaulicht
- Komplexität wird mit sogenannten **Fat-Wert** und **Design Tangles** berechnet



Mit freundlicher Genehmigung der Headway Software Technologies Ltd,
t/a Structure101.



Mit freundlicher Genehmigung der Headway Software Technologies Ltd,
t/a Structure101.

Vielen Dank für Ihre Aufmerksamkeit

Fragen?

Quellen

- Michael C. Feathers. *Working effectively with legacy code*. 10. print. Robert C. Martin series. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 2009. isbn : 978-0-13-117705-5
- Fernuni Hagen, Hrsg. *Refactoring-Tools in Eclipse* Eclipse. 10.05.2016.url: https://wiki.fernuni-hagen.de/eclipse/index.php/Refactoring-Tools_in_Eclipse
- Kai Spichale. *Kontinuierliche Architekturvalidierung: Auf die inneren Werte kommt es an*. Hrsg. von JAXenter. 2014.url:<https://jaxenter.de/kontinuierliche-architekturvalidierung-auf-die-inneren-werte-kommt-es-an-375>
- Michael Feathers. *Working Effectively With Legacy Code*.9.4.2002.url: <http://www.netobjectives.com/system/files/WorkingEffectivelyWithLegacyCode.pdf>
- QA Systems, Hrsg. *Code-Strukturen erfolgreich analysieren mit Structure101*. 24.03.2014. url: <http://www.qa-systems.de/produkte/structure101.html>
- Wikipedia, Hrsg. *Legacy code* - Wikipedia, the free encyclopedia. 12.06.2016.url: <https://en.wikipedia.org/w/index.php?oldid=713059641>
- Wikipedia, Hrsg. *PMD (Software)* .12.06.2016. url: <https://de.wikipedia.org/w/index.php?oldid=152729505>
- Structure 101, Hrsg. 20.6.2016. url: <http://structure101.com/products/#products=3>