Seminar Software Qualität im WS 2018/19

Vorbesprechung am 22. Juni 2018

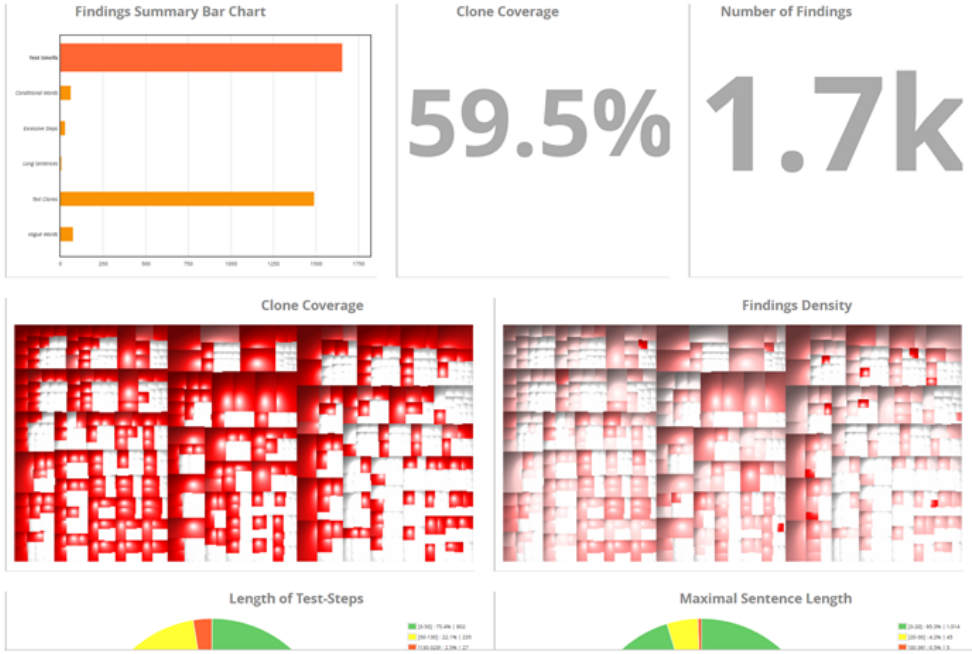
PD Dr. habil. Daniel Méndez

Dr. Maximilian Junker, Dr. Elmar Jürgens,

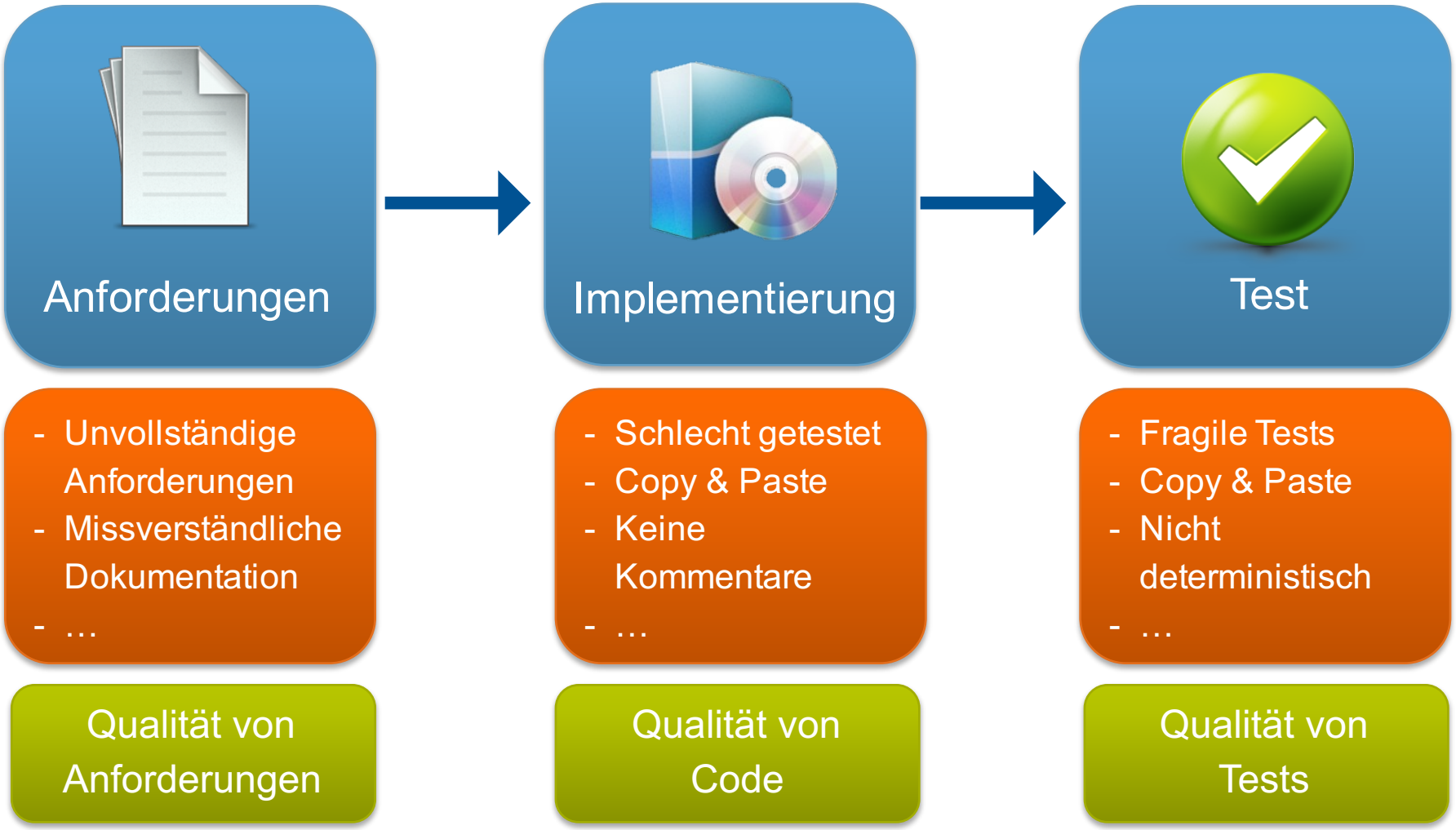
Dr. Benedikt Hauptmann,

Dr. Henning Femmer, Dr. Sebastian Eder,

Roman Haas, Daniel Veihelmann



Seminar Software Qualität



Teilnahme am Seminar

Mail an **daniel.mendez@tum.de**

bis **Sonntag, 1. Juli**

Inhalt

- Motivation für Seminarteilnahme u. Themen (<1 Seite)
- Stand im Studium
- Erfahrungen (Arbeit, Praktikum, Open-Source-Projekt, …)
- **Themenwünsche** (sortiert nach Priorität)

- Rückmeldung Mitte Juli über das Matching System
- Zuweisung Betreuer per Email durch uns

Ablauf

1. Ende Juli (nach Matching): Rückmeldung bei Betreuer und Festlegung des genauen Themas und Vorgehen
2. Semesteranfang (vstl. Mitte Oktober): Auftaktveranstaltung
   - Überblick Software Qualität
   - Literaturrecherche
   - Effektives Präsentieren
3. Vstl. Januar 2019*: Seminar / Vorträge

   (* Mitte Dezember auch möglich, je nach Präferenzen)
   - Blockveranstaltung: 3 Tage Vorträge mit Diskussion

*Terminfindungen via Doodle

Seminararbeit und Vortrag

- **Ausarbeitung:** max. 15 Seiten
- **Vortrag (Blockveranstaltung):** 25min + 20min Diskussion

Inhalt

- **Theorie**
- **Anwendung des Seminarthemas in der Praxis**
  (Ergebnisse, Erfahrungen, Probleme & Grenzen)

Abgaben

- Seminararbeit (vollständige Fassung): 2 Wochen vor Vortrag
- Probevortrag (verpflichtend): 2 Wochen vor Vortrag
- Seminararbeit (Endfassung): 1 Woche nach Vortrag

Sonstiges

- Bewertung / Zusammensetzung der Note: 50/50 (Ausarbeitung/Vortrag)
- Vorlagen werden zu Verfügung gestellt
- Anwesenheitspflicht bei Blockveranstaltungen

Themen (I)

Requirements Engineering und Anforderungsqualität

1. **Requirements Engineering in der Praxis.**
   Praktiken, Artefakte und typische Probleme (Daniel)

2. **Qualität von Anforderungen.** Welche Qualitätsmerkmale für Anforderungen existieren und welche Relevanz haben diese in Projekten? (Daniel)

Themen (II)

Qualitätssicherung von Anforderungen

3. **Automatische Qualitätssicherung von Anforderungen**:
   Eindeutigkeit – Möglichkeiten und Grenzen automatischer Erkennung
   (Henning)

4. **Automatische Qualitätssicherung von Anforderungen**:
   Faktoren von Lesbarkeit
   (Henning)

Automatische Qualitätssicherung von Anforderungen

Eindeutigkeit – Möglichkeiten und Grenzen automatischer Erkennung

- Anforderungen werden fast ausschließlich (>90%) in natürlicher Sprache dokumentiert
- Grund: Natürliche Sprache ermöglicht es allen Stakeholdern gemeinsam in Dokumenten zu schreiben und zu lesen.
- Nachteil: Natürliche Sprache ist inhärent mehrdeutig.
- Beispiel:

> "The product shall show the weather for the next 24 hours."

Frage: Kann man diese Mehrdeutigkeiten automatisch erkennen?
Betreuer: Henning Femmer

Automatische Qualitätssicherung von Anforderungen

Faktoren von Lesbarkeit

- Anforderungen werden fast ausschließlich (>90%) in natürlicher Sprache dokumentiert
- Problem: Requirements Engineers sind nicht darauf trainiert schnell und einfach lesbare Anforderungen zu schreiben.
- Beispiel:

> Dies bedeutet, das E-Mails, die nicht im Ordner "gesendet" (oder irgendeinem Unterordner) gespeichert wurden, in die entsprechende Ordnerstruktur im privaten E-Mailverzeichnis verschoben werden.

Frage: Kann man Lesbarkeit automatisch bewerten oder Verletzungen erkennen?

Betreuer: Henning Femmer

Themen (III)

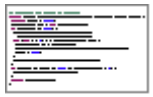
Qualität von Code

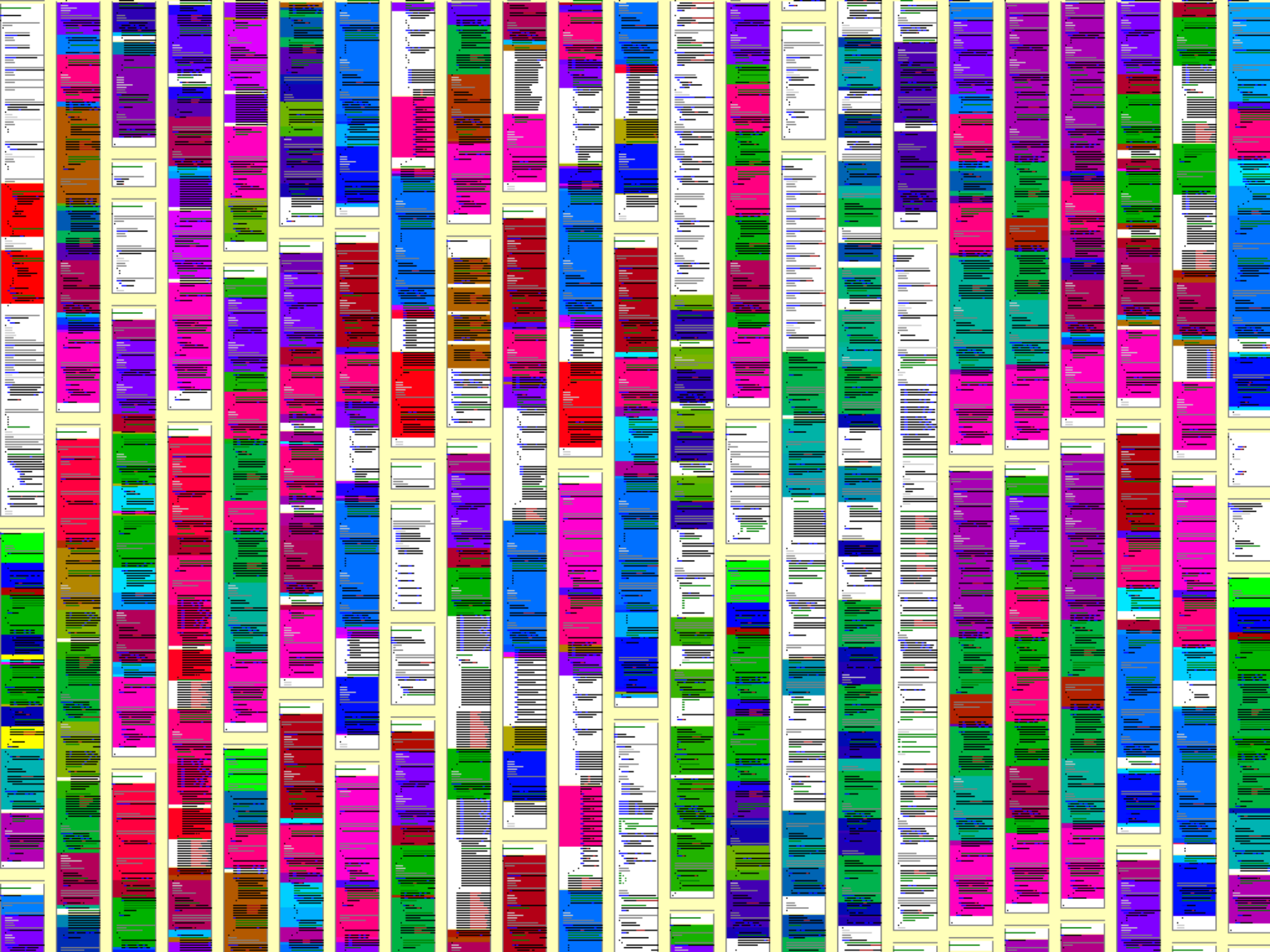
5. **Statische Analyse:** Automatisches Auffinden von Bugs in Quellcode. (Sebastian)
6. **Clone Detection:** Wo ist kopierter Quelltext im System? (Elmar)
7. **Test Impact Analyse** (Elmar)

```java
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
   boolean found = false;
   StringBuffer res = new StringBuffer();
   if (elements != null) {
      Index.getInstance().setCurrentRenderer(
            FlatReferenceRenderer.getInstance());
      for (int i = 0; i < elements.length; i++) {
         ModelElement el = elements[i];
         res.append(showElementLink(el)).append(HTML.LINE_BREAK);
         found = true;
      }
      Index.getInstance().resetCurrentRenderer();
   }
   if (!found && nomsg != null && nomsg.length() > 0) {
      res.append(HTML.italics(nomsg));
   }
   return res.toString();
}
```

```java
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
   boolean found = false;
   StringBuffer res = new StringBuffer();
   if (elements != null) {
      Index.getInstance().setCurrentRenderer(
            FlatReferenceRenderer.getInstance());
      for (int i = 0; i < elements.length; i++) {
         ModelElement el = elements[i];
         res.append(showElementLink(el)).append(HTML.LINE_BREAK);
         found = true;
      }
      Index.getInstance().resetCurrentRenderer();
   }
   if (!found && nomsg.length() > 0) {
      res.append(HTML.italics(nomsg));
   }
   return res.toString();
}
```

```
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
   boolean found = false;
   StringBuffer res = new StringBuffer();
   if (elements != null) {
      Index.getInstance().setCurrentRenderer(
            FlatReferenceRenderer.getInstance());
      for (int i = 0; i < elements.length; i++) {
         ModelElement el = elements[i];
         res.append(showElementLink(el)).append(HTML.LINE_BREAK);
         found = true;
      }
      Index.getInstance().resetCurrentRenderer();
   }
   if (!found && nomsg != null && nomsg.length() > 0) {
      res.append(HTML.italics(nomsg));
   }
   return res.toString();
}
```

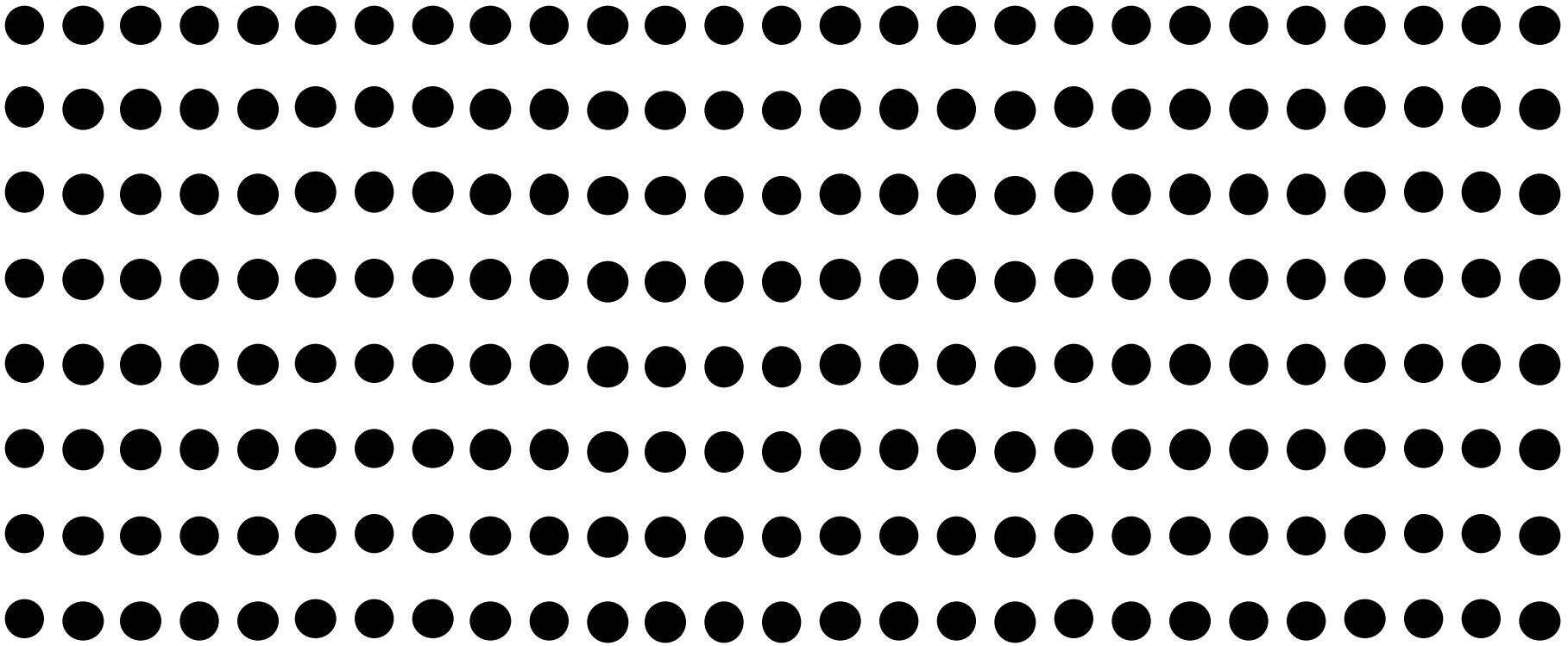


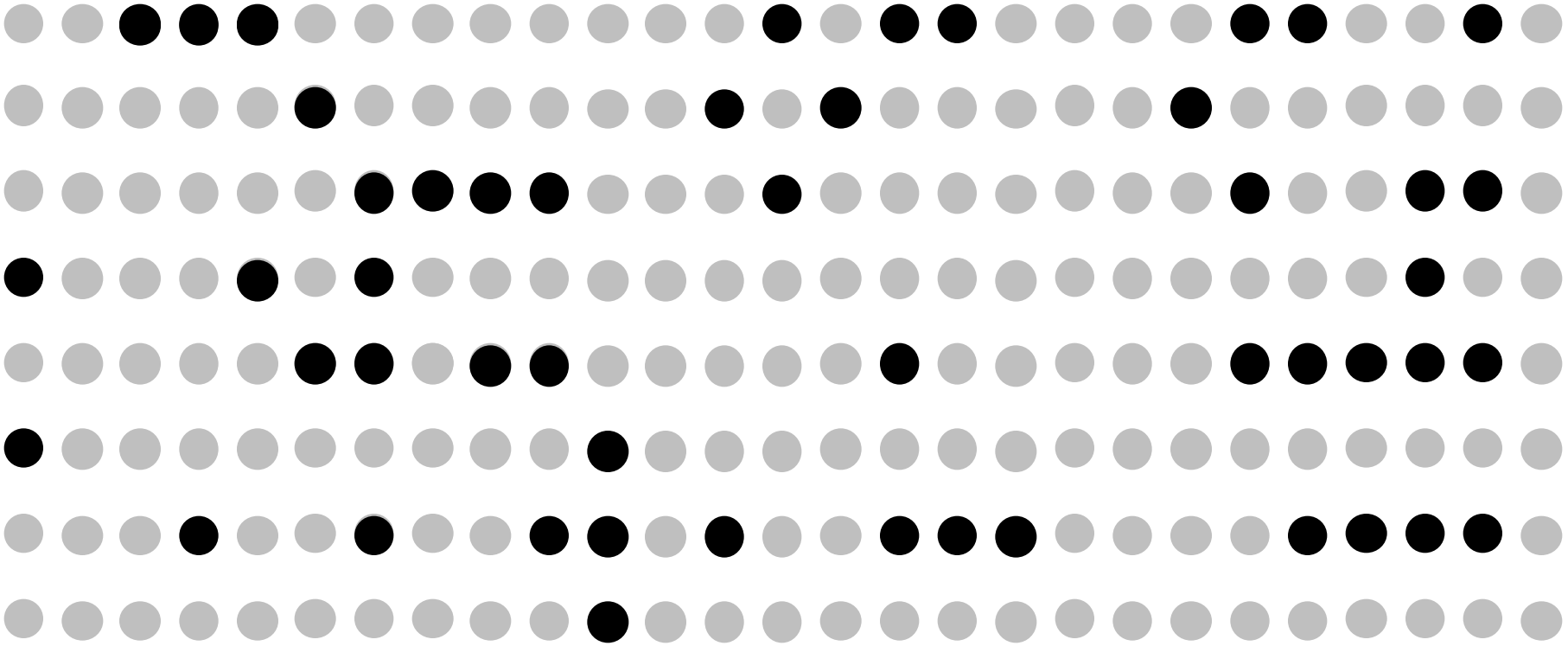


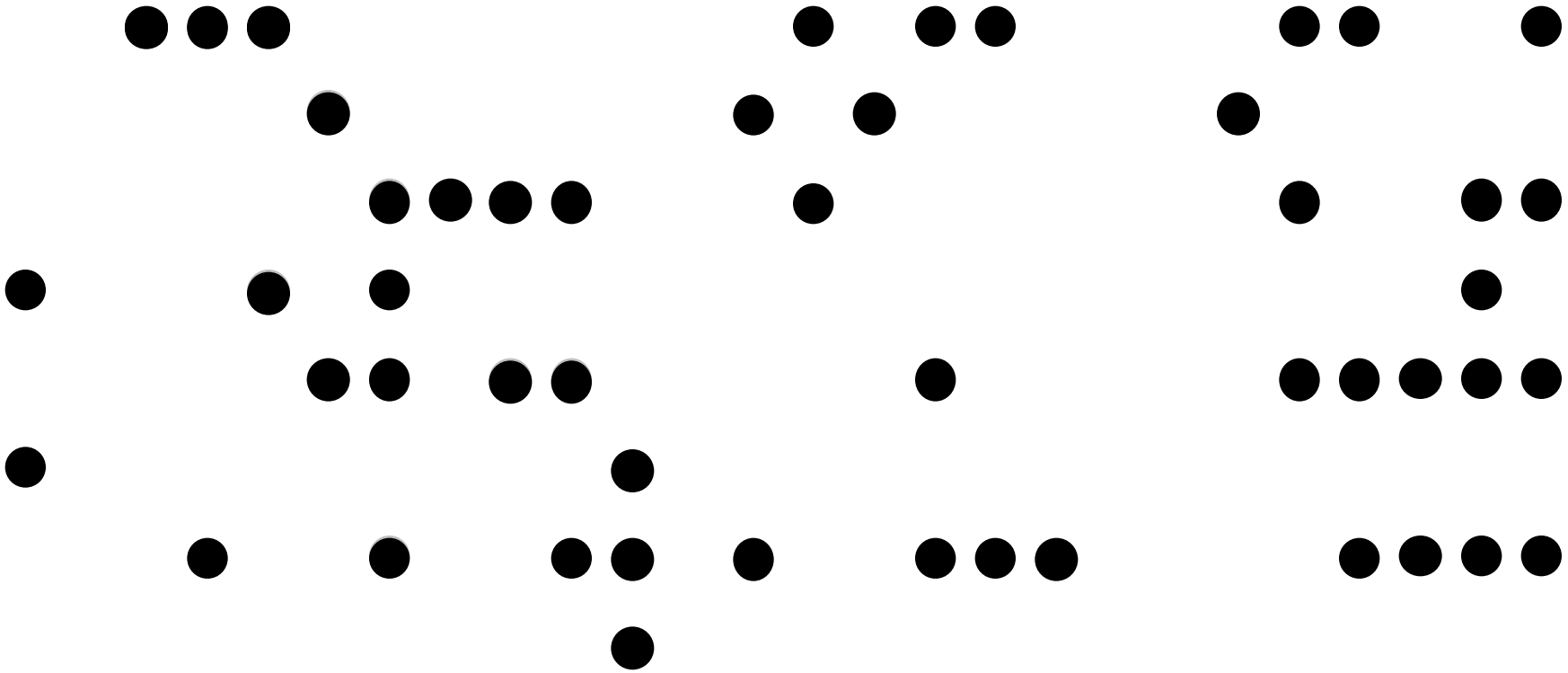
Themen (III)

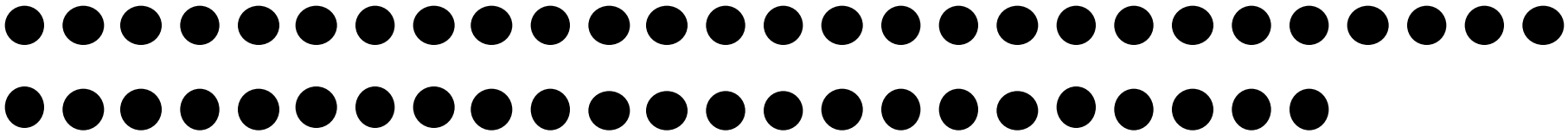
Qualität von Code

5. **Statische Analyse:** Automatisches Auffinden von Bugs in Quellcode. (Sebastian)
6. **Clone Detection:** Wo ist kopierter Quelltext im System? (Elmar)
7. **Test Impact Analyse** (Elmar)







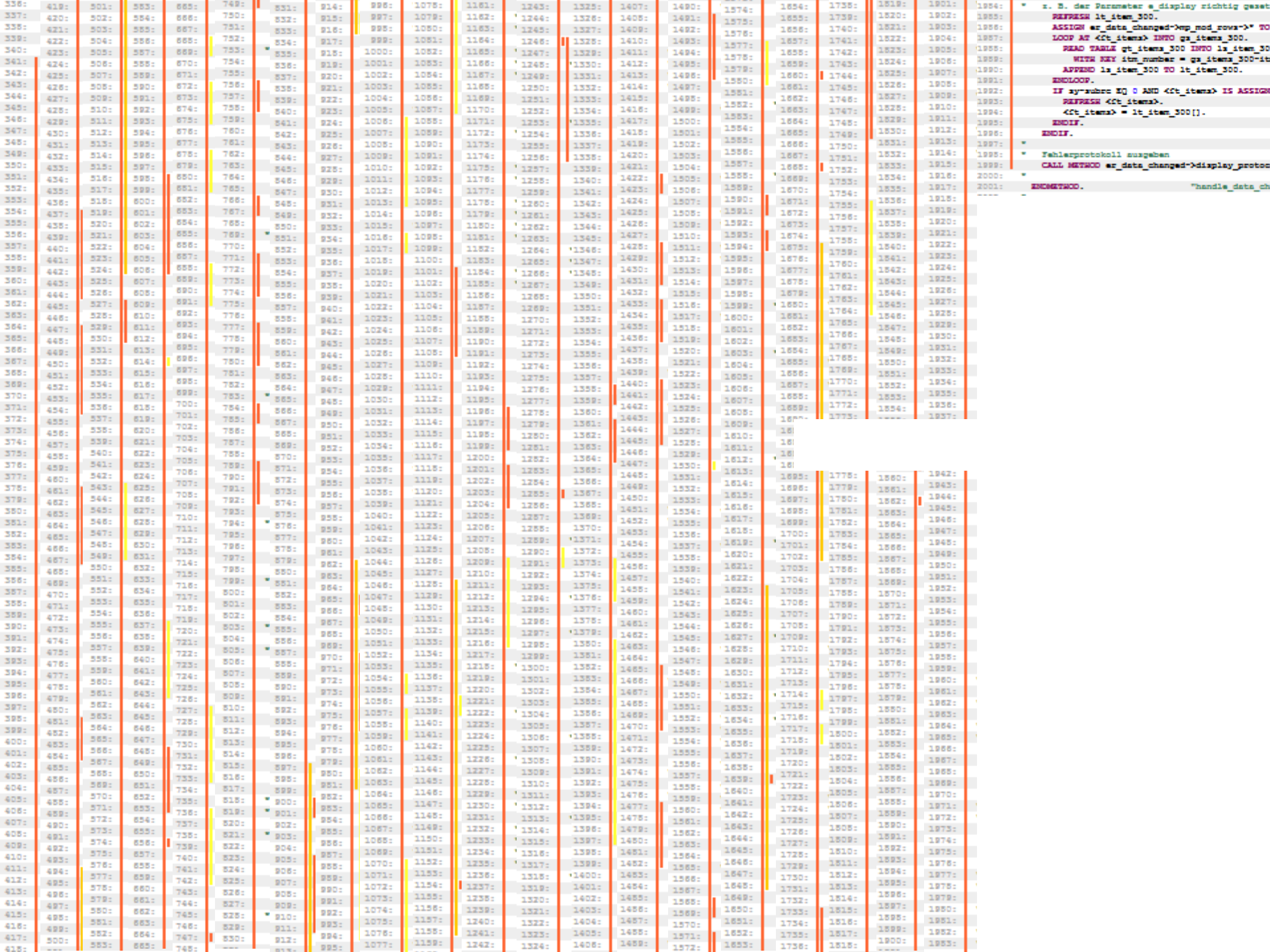


Themen (IV)

Qualität von Code

8. **Refactoring Empfehlungen:** Automatische Generierung von Verbesserungsvorschlägen für Code.
   (Roman Haas)

9. **Code-Zentralität:** Was sind in diesem Softwaresystem die wichtigsten Klassen?
   (Roman Haas)

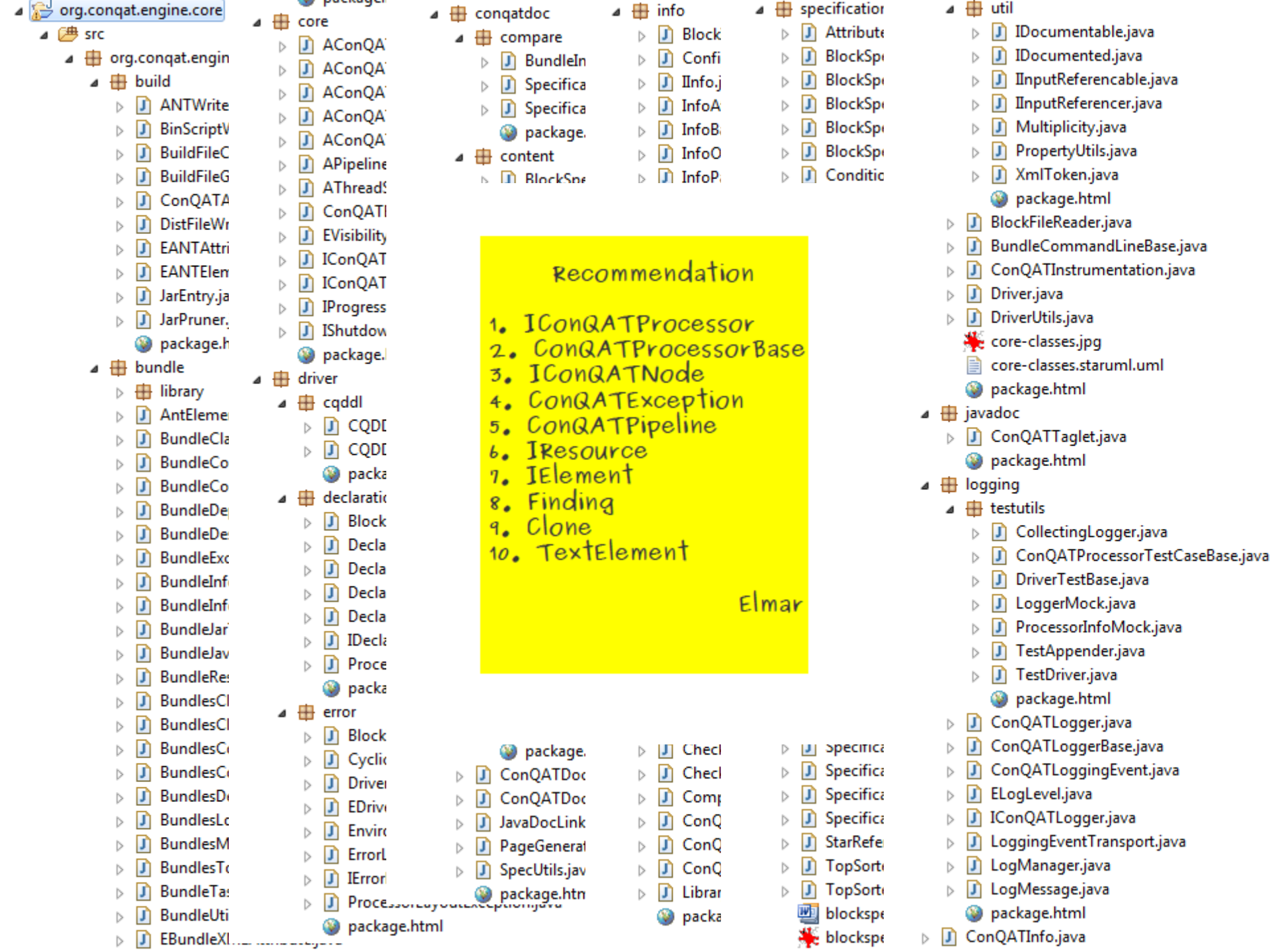


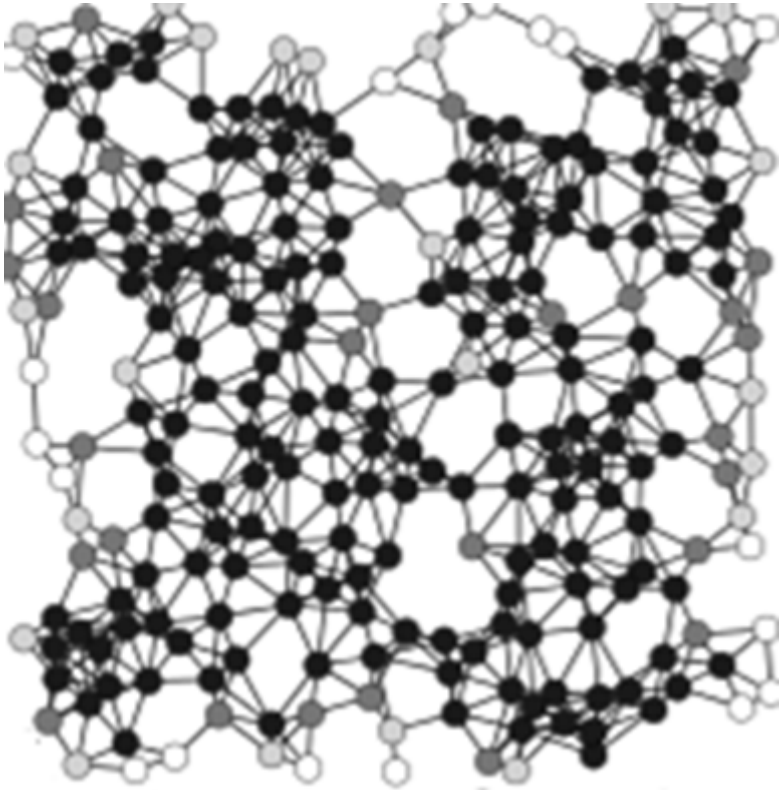
Themen (IV)

Qualität von Code

8. **Refactoring Empfehlungen:** Automatische Generierung von Verbesserungsvorschlägen für Code.
   (Roman Haas)

9. **Code-Zentralität:** Was sind in diesem Softwaresystem die wichtigsten Klassen?
   (Roman Haas)


org.conqat.engine.core
src
org.conqat.engin
build
ANTWrite
BinScriptW
BuildFileC
BuildFileG
ConQATA
DistFileWr
EANTAttri
EANTElem
JarEntry.ja
JarPruner,
package.h
bundle
library
AntEleme
BundleCla
BundleCo
BundleCo
BundleDe
BundleDe
BundleExc
BundleInf
BundleInf
BundleJar
BundleJav
BundleRes
BundlesCl
BundlesCl
BundlesC
BundlesC
BundlesD
BundlesLc
BundlesM
BundlesTc
BundleTas
BundleUti
EBundleXl
core
AConQA
AConQA
AConQA
AConQA
AConQA
APipeline
AThreadS
ConQATI
EVisibility
IConQAT
IConQAT
IProgress
IShutdow
package.
driver
cqddl
CQDD
CQDD
packa
declaratic
Block
Decla
Decla
Decla
Decla
IDecla
Proce
packa
error
Block
Cyclic
Driver
EDrive
Envirc
ErrorL
IError
Proces
package.html
conqatdoc
compare
BundleIn
Specifica
Specifica
package.
content
BlockSpe
package.
ConQATDoc
ConQATDoc
JavaDocLink
PageGenerat
SpecUtils.jav
package.htn
info
Block
Confi
IInfo.j
InfoA
InfoB
InfoO
InfoP
Checl
Checl
Comp
ConQ
ConQ
ConQ
Librar
packa
specificatior
Attribute
BlockSp
BlockSp
BlockSp
BlockSp
BlockSp
Conditic
Specifica
Specifica
Specifica
Specifica
StarRefe
TopSort
TopSort
blockspe
blockspe
util
IDocumentable.java
IDocumented.java
IInputReferencable.java
IInputReferencer.java
Multiplicity.java
PropertyUtils.java
XmlToken.java
package.html
BlockFileReader.java
BundleCommandLineBase.java
ConQATInstrumentation.java
Driver.java
DriverUtils.java
core-classes.jpg
core-classes.staruml.uml
package.html
javadoc
ConQATTaglet.java
package.html
logging
testutils
CollectingLogger.java
ConQATProcessorTestCaseBase.java
DriverTestBase.java
LoggerMock.java
ProcessorInfoMock.java
TestAppender.java
TestDriver.java
package.html
ConQATLogger.java
ConQATLoggerBase.java
ConQATLoggingEvent.java
ELogLevel.java
IConQATLogger.java
LoggingEventTransport.java
LogManager.java
LogMessage.java
package.html
ConQATInfo.java
Recommendation
1. IConQATProcessor
2. ConQATProcessorBase
3. IConQATNode
4. ConQATException
5. ConQATPipeline
6. IResource
7. IElement
8. Finding
9. Clone
10. TextElement
Elmar



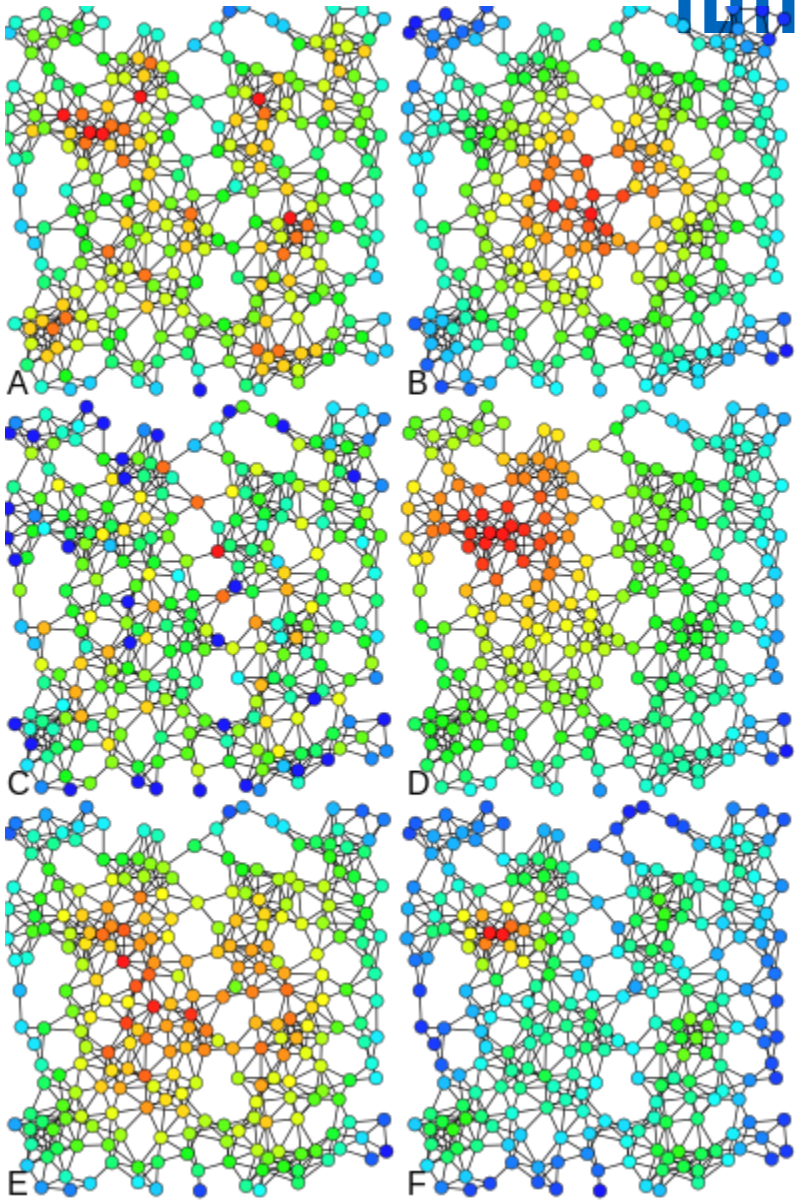


Bild: http://en.wikipedia.org/wiki/File:Centrality.svg

Themen (V)

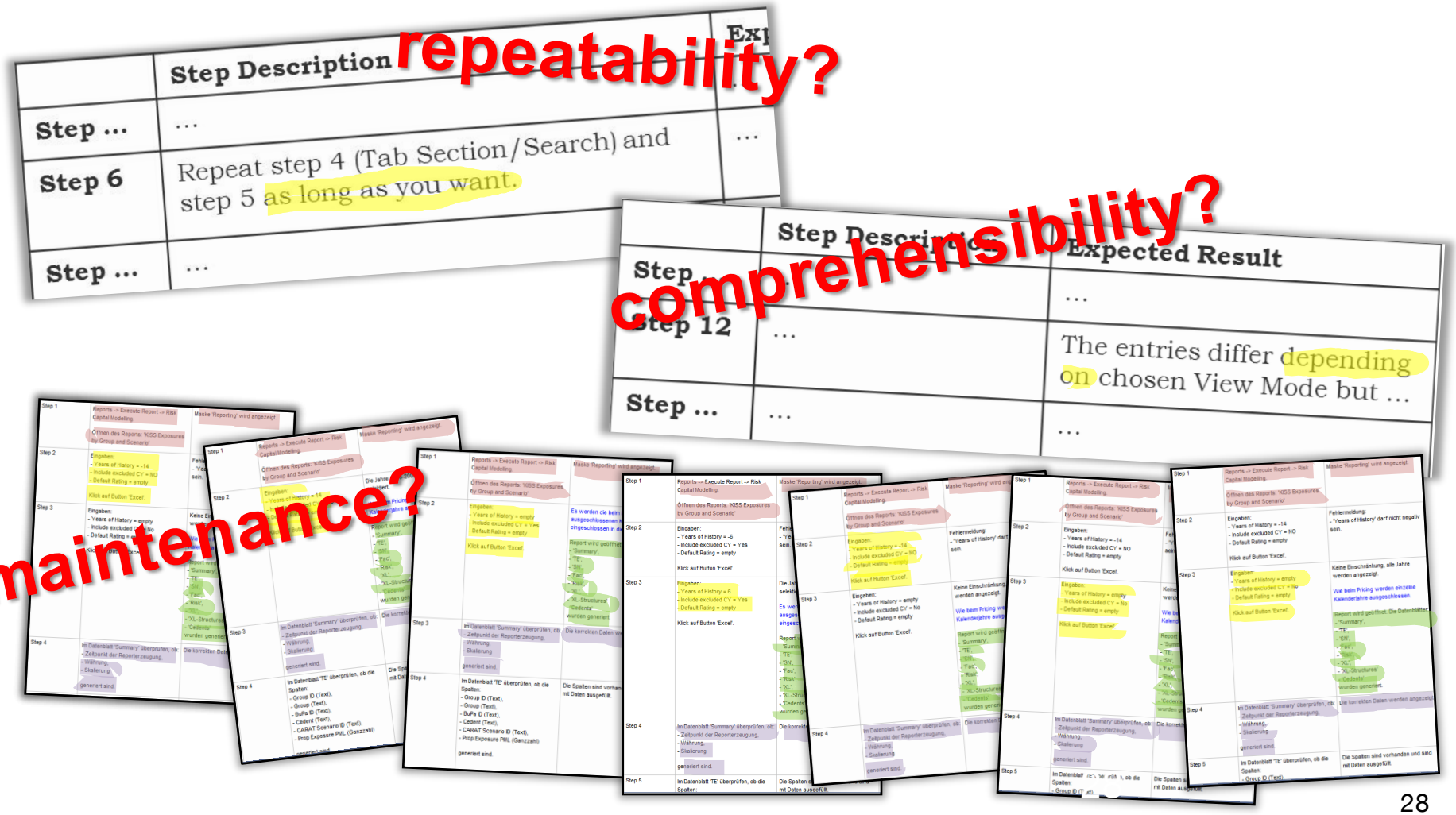
Qualität von Tests

10. **Natural Language Test Smells**: Qualitätsdefekte in manuellen Tests automatisch erkennen.
    (Benedikt)

11. **Warum zerbrechen meine automatisierten GUI-Tests?** Statische Analyse von automatisierten Tests.
    (Benedikt)

12. **Bessere Tests durch Testfallgenerierung?**
    (Maximilian)

Natural Language Test Smells: Qualitätsdefekte in manuellen Tests automatisch erkennen.



Warum zerbrechen meine automatisierten GUI-Tests?

Statische Analyse von automatisierten Tests.

Automated Test 01 – Login Dialog			
1.	Open application	C:\programs\keepass\2.0\keepass.exe	
2.	Validate	Windows is visible	“Open Database – New Datbase.kdbx”
3.	Enter value	“Master Password”	“123mysecret$”
4.	Click	Button	“Cancel”
	…	…	…

Themen (V)

Qualität von Tests

10. **Natural Language Test Smells**: Qualitätsdefekte in manuellen Tests automatisch erkennen.
    (Benedikt)

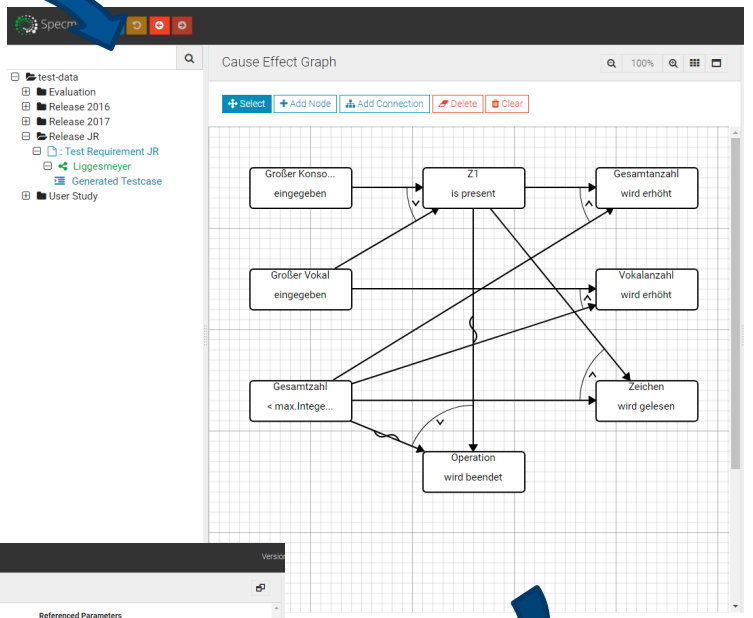
11. **Warum zerbrechen meine automatisierten GUI-Tests?** Statische Analyse von automatisierten Tests.
    (Benedikt)

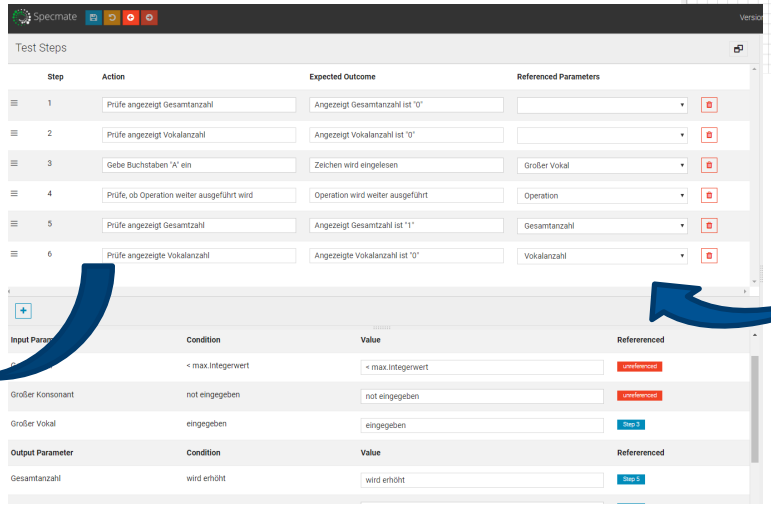
12. **Bessere Tests durch Testfallgenerierung?**
    (Maximilian)

Bessere Tests durch Testfallgenerierung?

Description:

Die Operation "Zähle Zeichen" liest Zeichen von der Tastatur, solange große Konsonanten oder große Vokale eingegeben werden sowie die Gesamtzahl kleiner ist als der Maximalwert des Datentyps integer.
Ist ein gelesenes Zeichen ein großer Konsonant oder Vokal, so wird die Gesamtzahl um eins erhöht. Falls das eingelesene Zeichen ein großer Vokal ist, so wird auch die Vokalanzahl um eins erhöht.



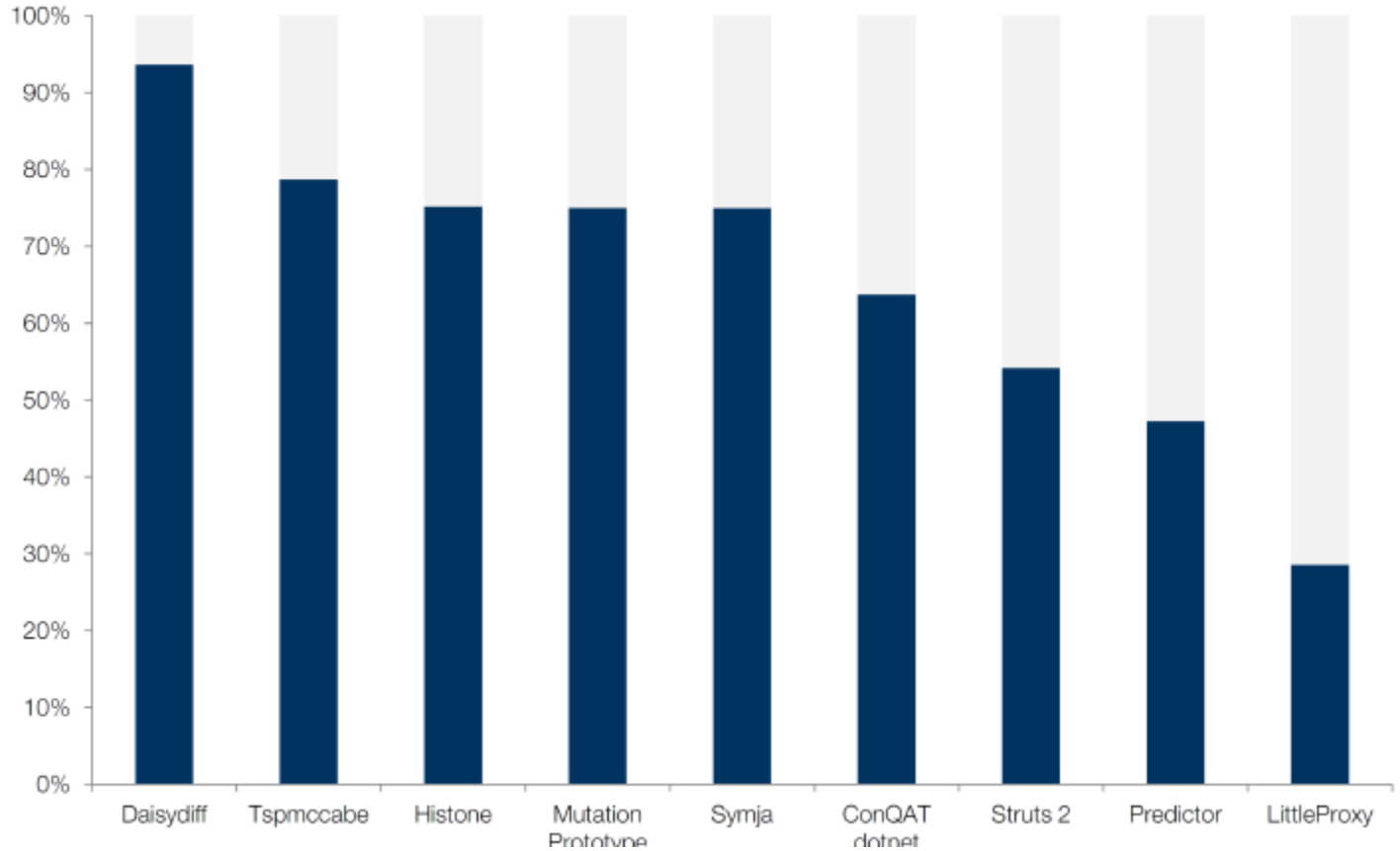


Themen (VI)

Qualität von Tests

13. **Qualitätssicherung von Tests durch Mutation Testing**
    (Daniel Veihelmann)

14. **Test Gap Analyse:** Werden Systemänderungen auch wirklich von Tests erfasst?
    (Daniel Veihelmann)

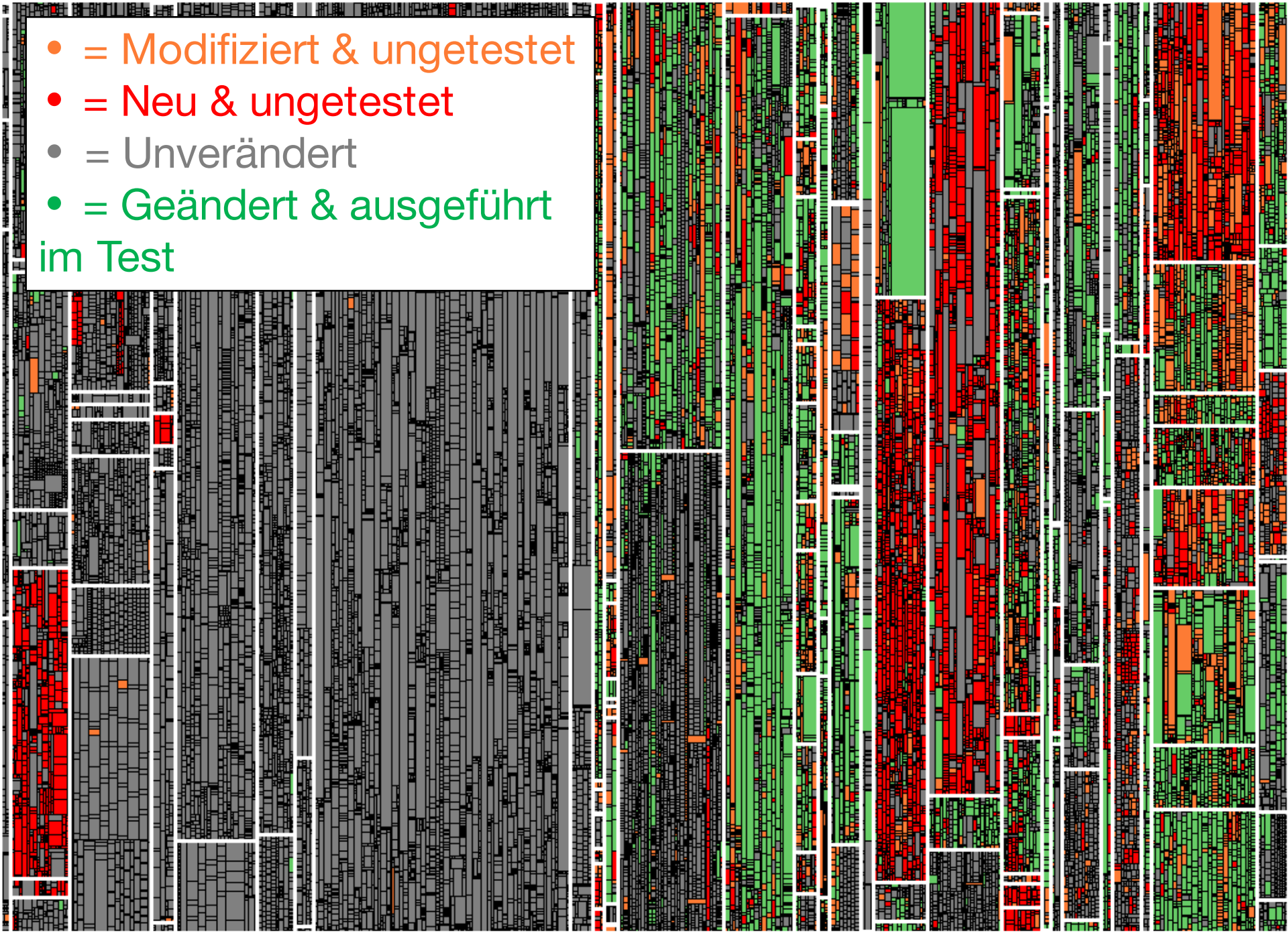

100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Daisydiff
Tspmccabe
Histone
Mutation Prototype
Symja
ConQAT dotnet
Struts 2
Predictor
LittleProxy

Themen (VI)

Qualität von Tests

13. **Qualitätssicherung von Tests durch Mutation Testing**
    (Daniel Veihelmann)

14. **Test Gap Analyse:** Werden Systemänderungen auch wirklich von Tests erfasst?
    (Daniel Veihelmann)

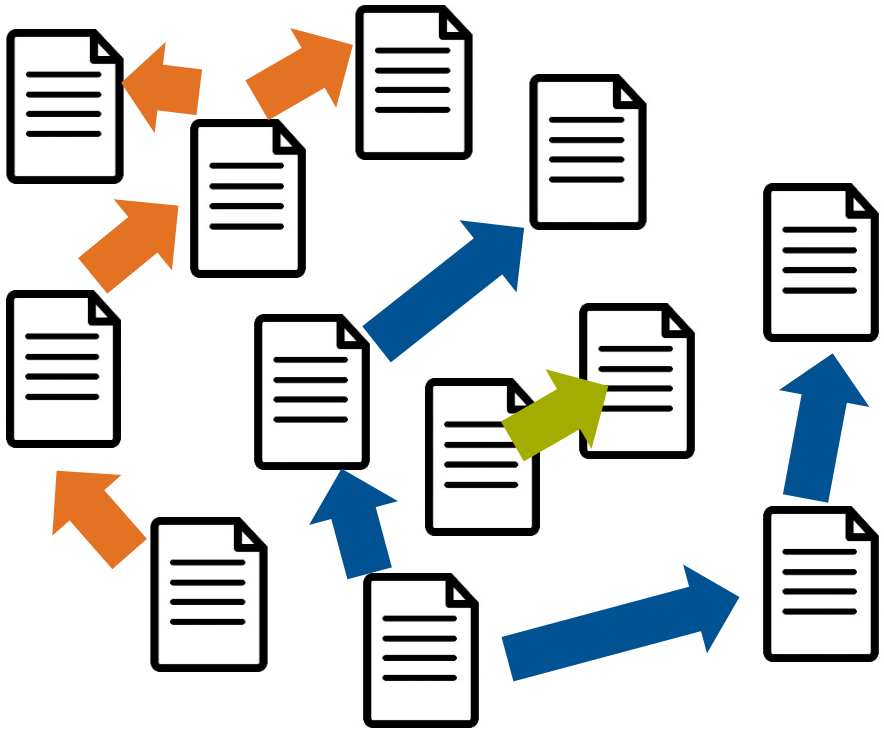

• = Modifiziert & ungetestet
• = Neu & ungetestet
• = Unverändert
• = Geändert & ausgeführt
im Test

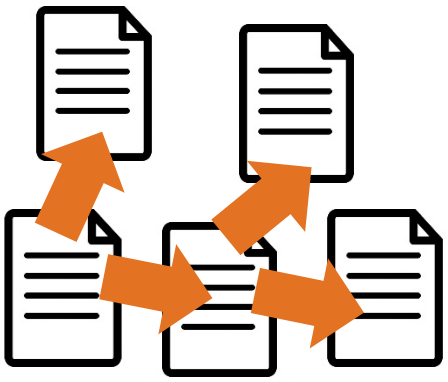
Themen (VII)

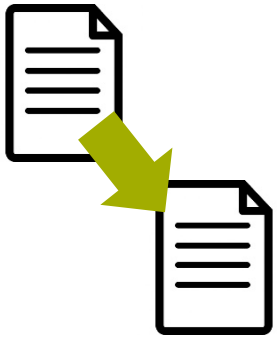
Artificial Intelligence in Requirements Engineering

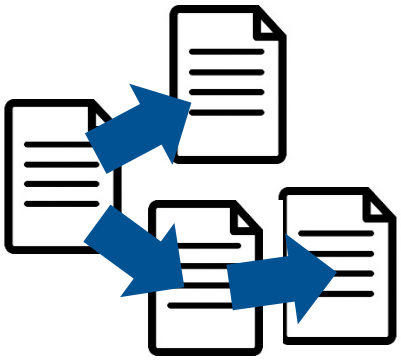
15. **Tracing zwischen Requirements**: Welche Requirements sind ähnlich?
    (Sebastian)

16. **Glossary Extraction:** Was sind die wichtigsten Begriffe in meinen Requirements?
    (Maximilian)







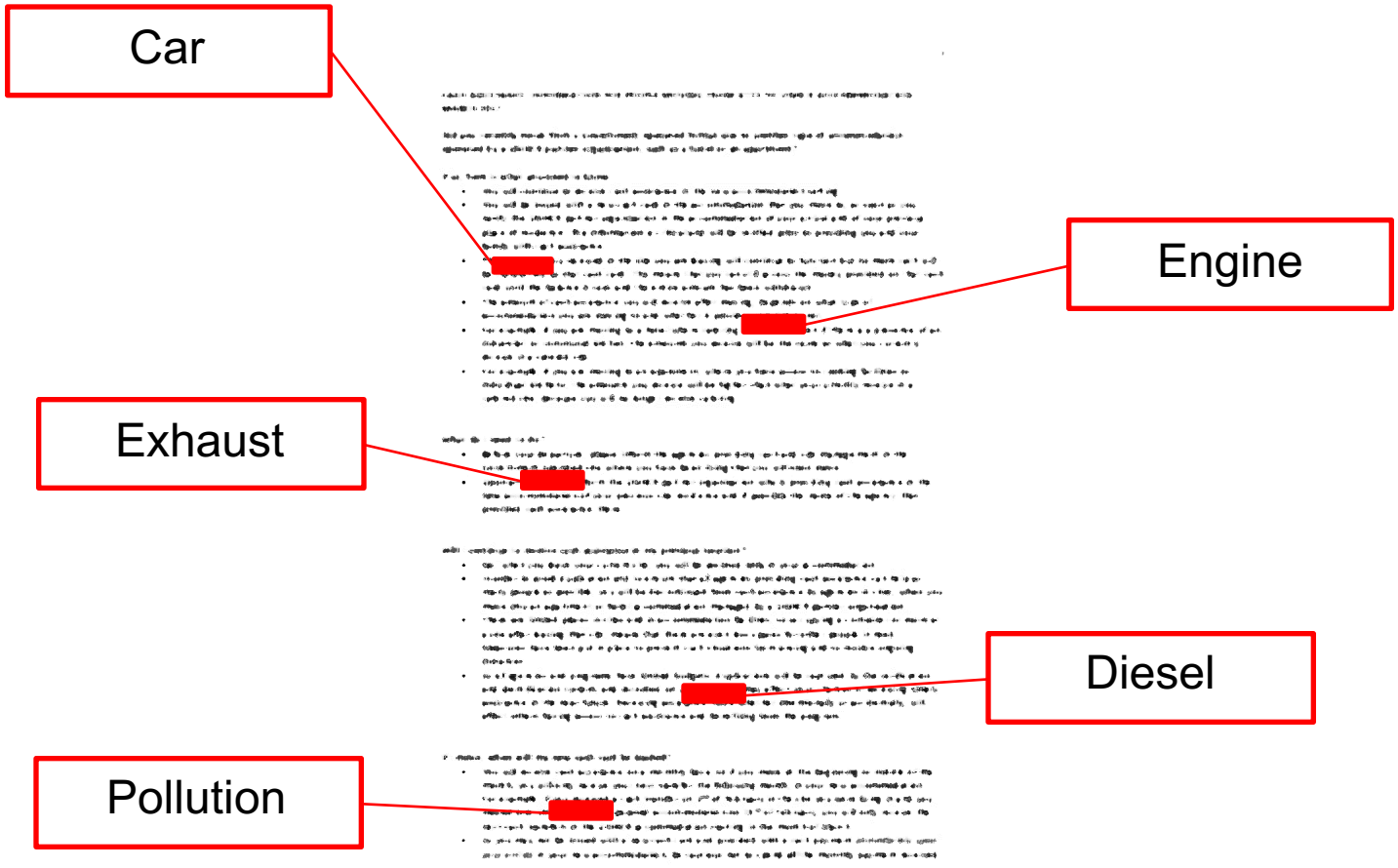


Themen (VII)

Artificial Intelligence in Requirements Engineering

15. **Tracing zwischen Requirements**: Welche Requirements sind ähnlich?
    (Sebastian Eder)

16. **Glossary Extraction:** Was sind die wichtigsten Begriffe in meinen Requirements?
    (Maximilian Junker)


Car
Engine
Exhaust
Diesel
Pollution

Fragen? Anmerkungen?