Wie man Clean Code Development in Brownfield Projekten betreibt ohne den Verstand zu verlieren

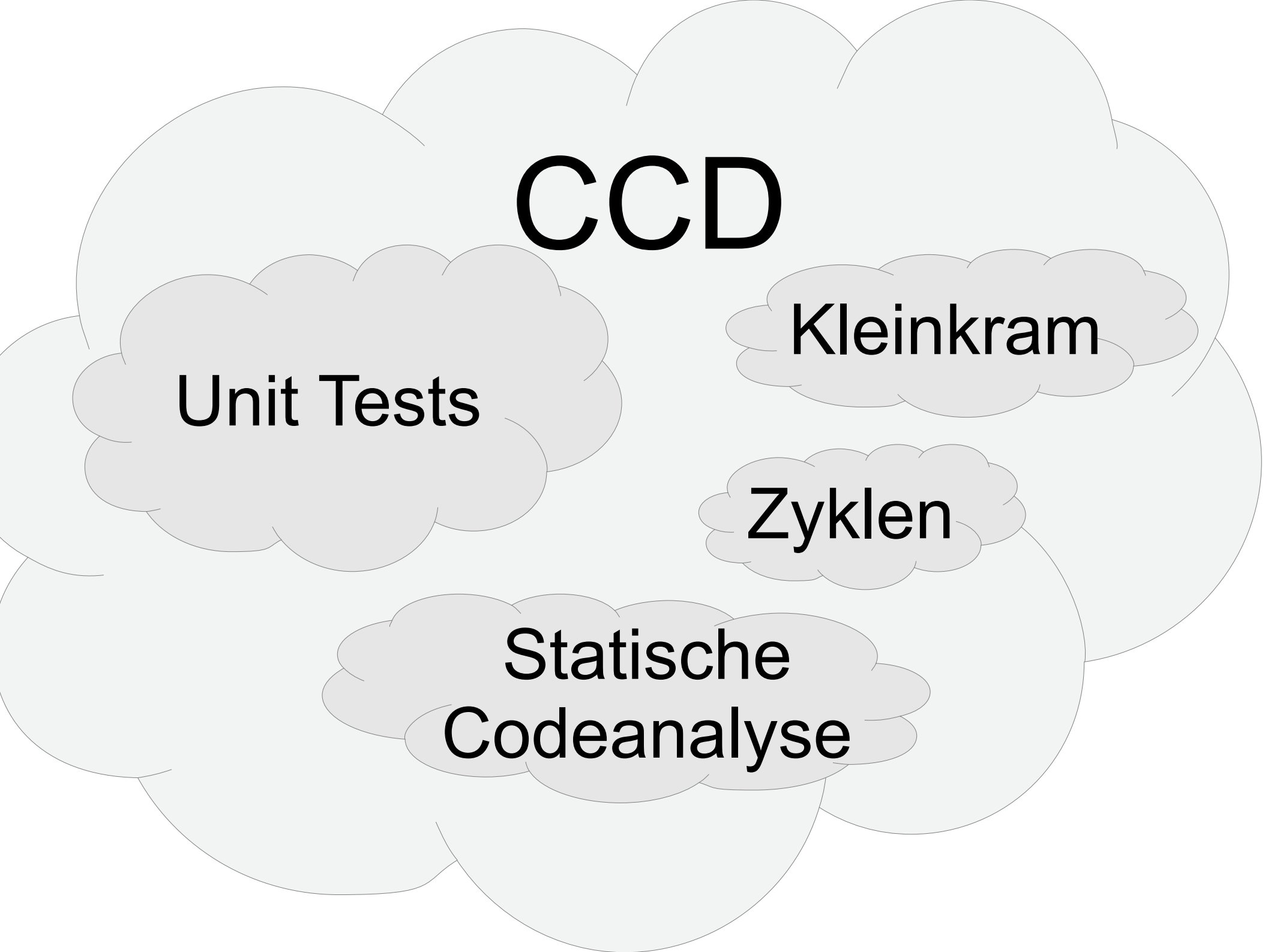
Jens Nerche

Leiter Anwendungsentwicklung

Kontext E GmbH

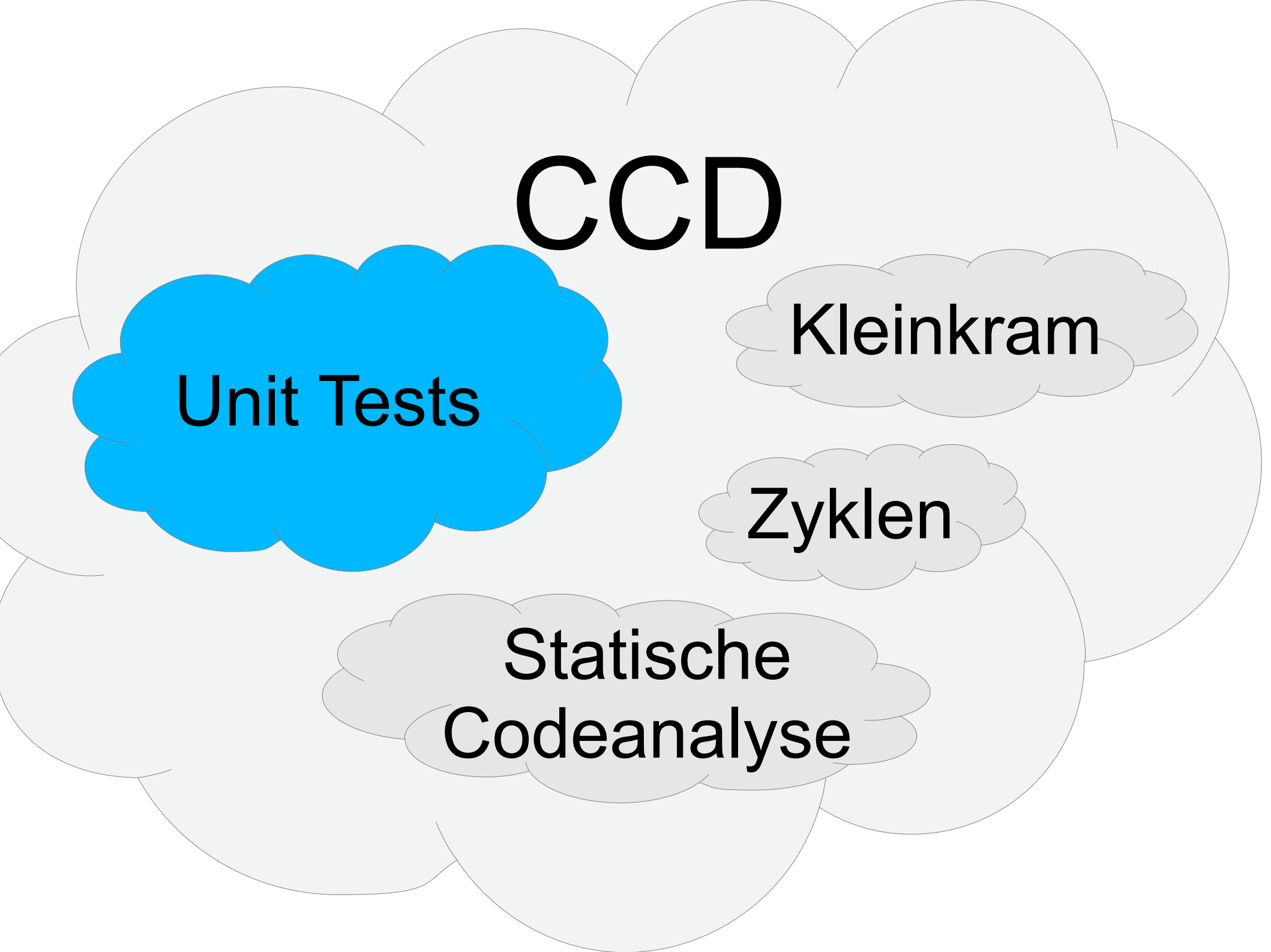
www.kontext-e.de

Email	j.nerche@kontext-e.de
Twitter	@jensnerche
Blog	http://wp.me/p3Ieb2-Z


CCD
Unit Tests
Kleinkram
Zyklen
Statische
Codeanalyse

Der psychologische Aspekt

Frust	Motivation
Aufgeben	Durchhalten
Erschrecken	Mut


CCD
Unit Tests
Kleinkram
Zyklen
Statische
Codeanalyse

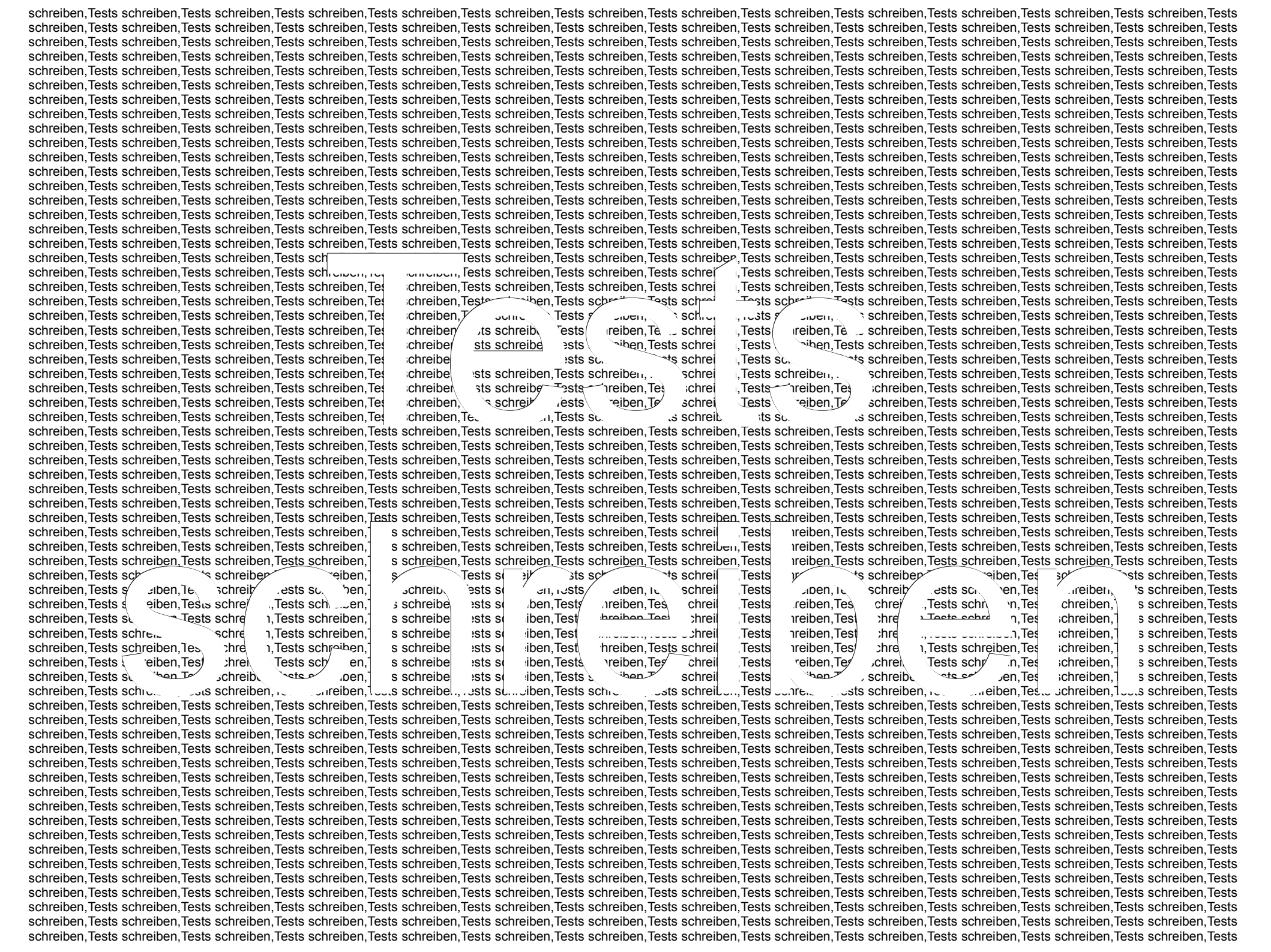
Brownfield!

Testabdeckung?

Aaaaaaaarrrrrghhhh!!

(Was nun?)

Mindesttestabdeckung
1%


Tests schreiben

Gähn! Frust! Nutzen?

Ha! Nur für geänderte Dateien gilt...

Refactorings?

- Rename
- Remove Unused Parameter
- Introduce Parameter
- Move Class

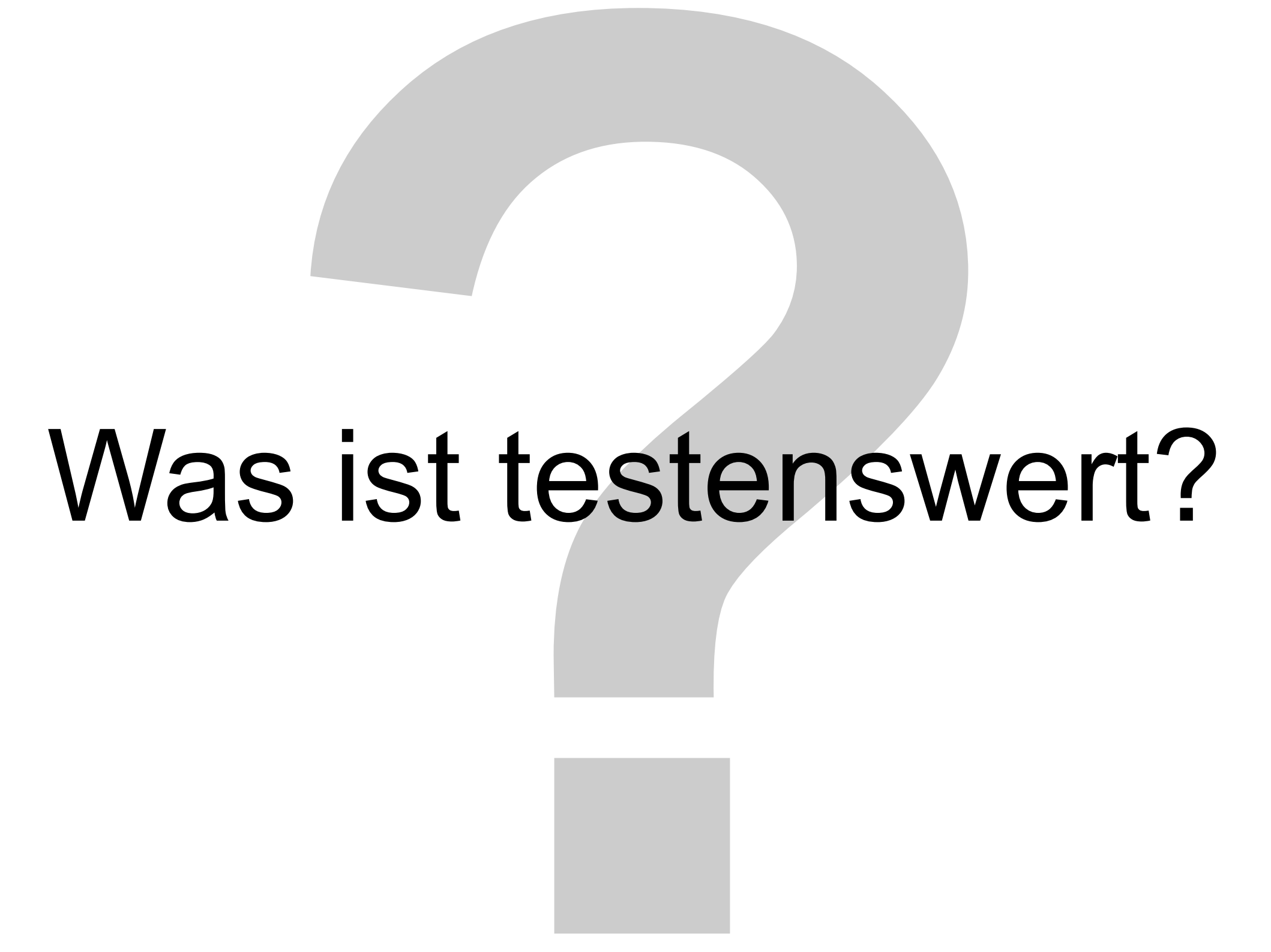
Ätsch! Ich kann ja Properties, equals und hash ganz leicht testen!

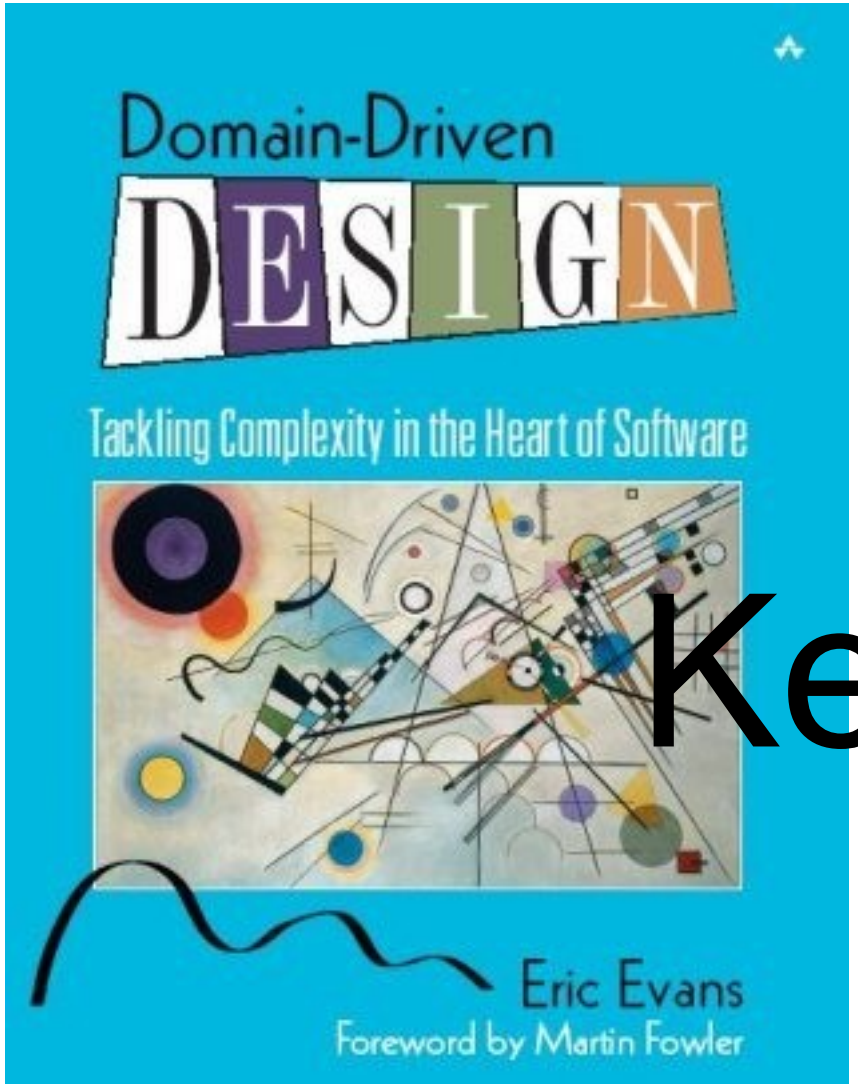
Nutzen?

Dann eben doch manuell prüfen!

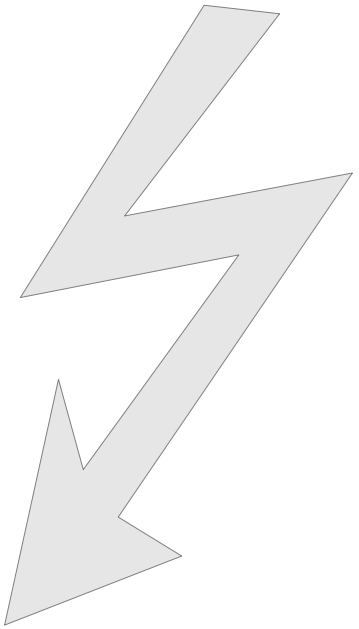
Das. Geht. Nicht. Lange. Gut.

Also wie jetzt?


Was ist testenswert?


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

Kerndomäne



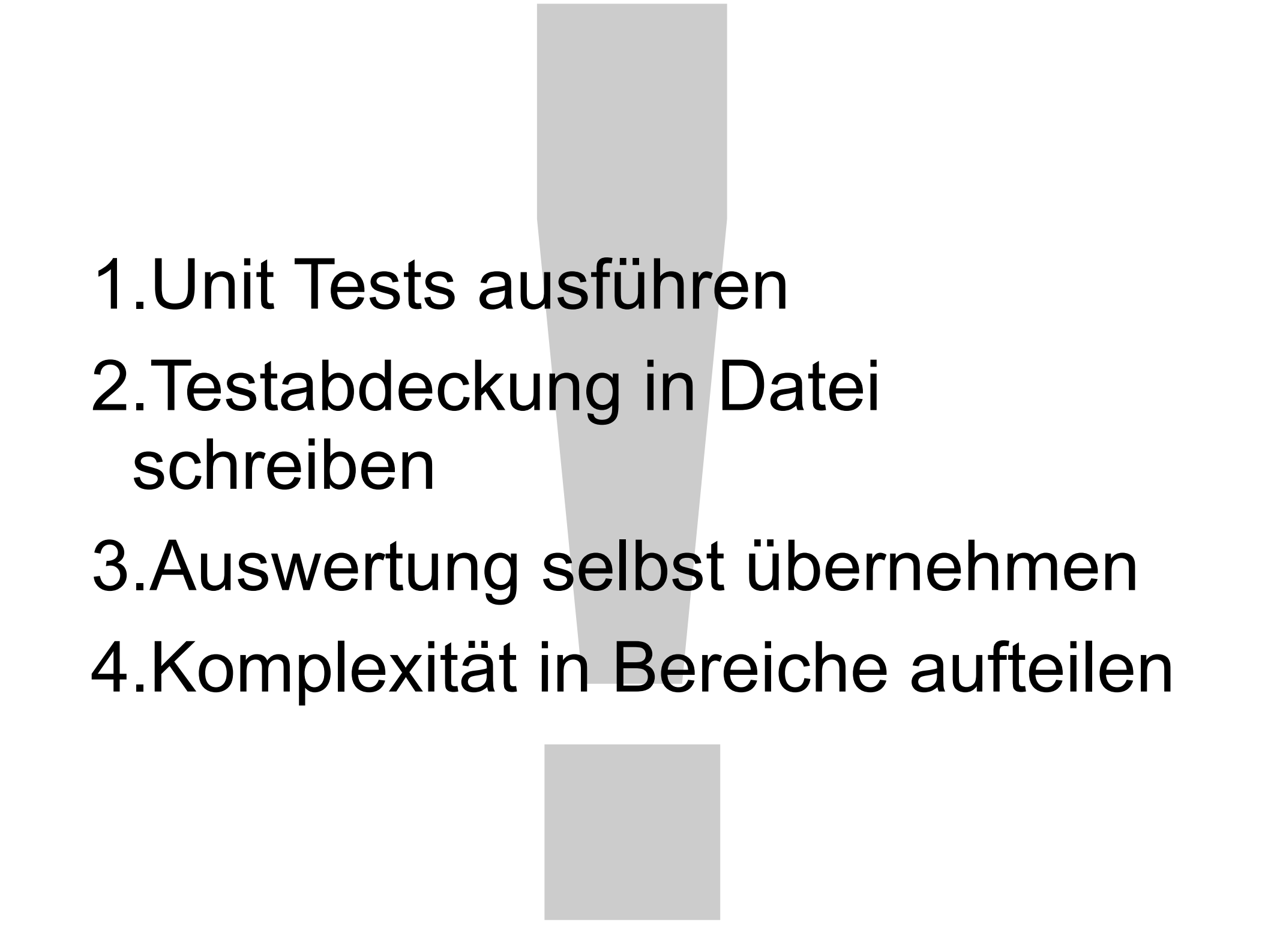
Risiko:

häufige Änderungen

VCS

komplexer Code

McCabe


1. Unit Tests ausführen
2. Testabdeckung in Datei schreiben
3. Auswertung selbst übernehmen
4. Komplexität in Bereiche aufteilen

auf Methodenebene - nicht Durchschnitt der Klasse o.ä.

eine Anzahl von Methoden erlauben, die nicht die Mindesttestabdeckung erreichen

Pakete/Namensräume definieren, die „sauber“ sind

von Iteration zu Iteration verbessern

Beispiel

trivial 1, niedrig 2..3, mittel 4..5, hoch 6..8, sehr hoch >8

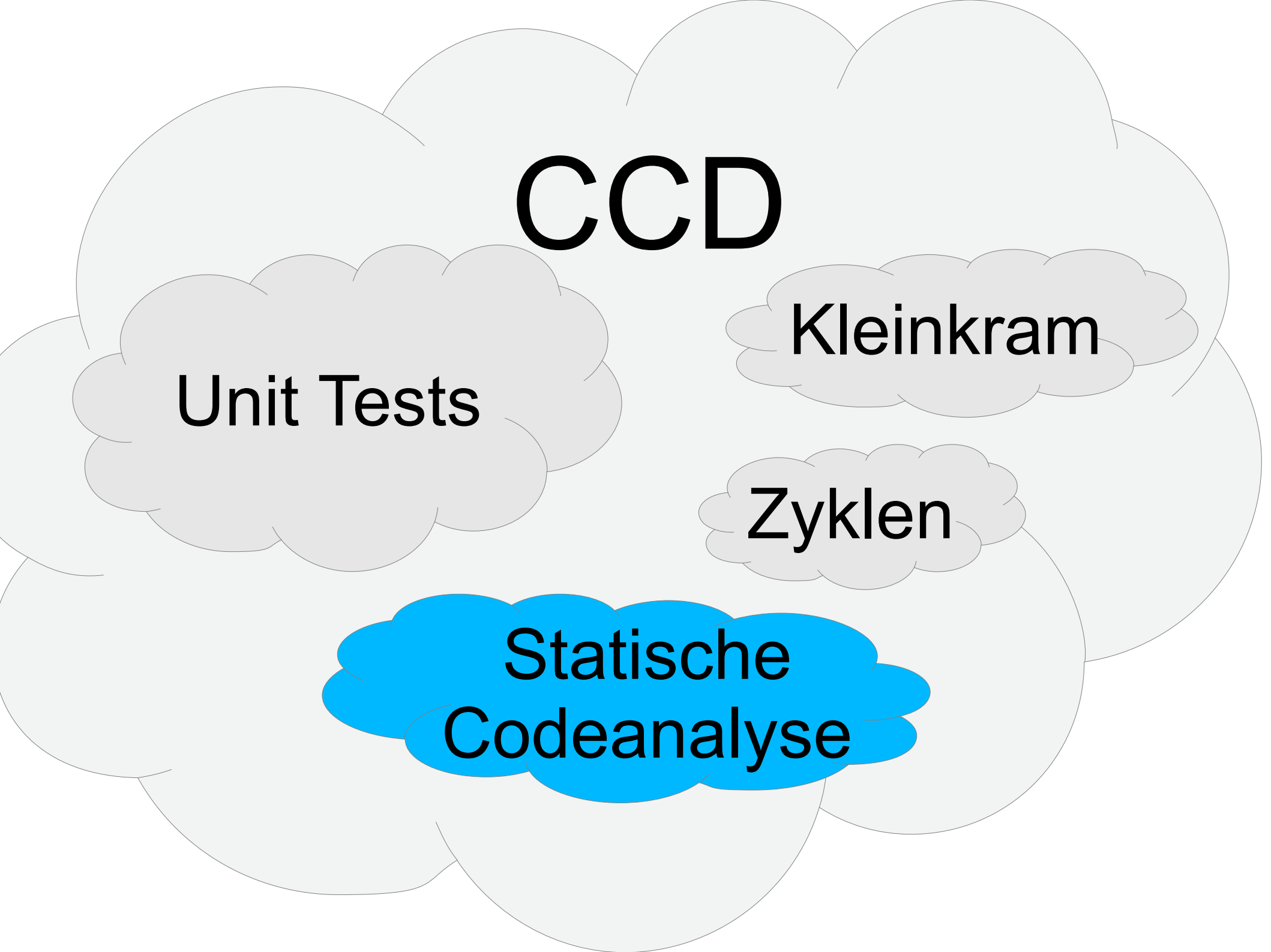
Mindesttestabdeckung pro Bereich

Komplexität > 10 ➡ Abdeckung > 80%

```
class CoverageCheckerConfiguration {
  static def rangeOfLowComplexity    = 2..3
  static def rangeOfMediumComplexity = (rangeOfLowComplexity.last()+1)..5

  static def minBranchCoverageForHighComplexity    = 90
  static def minBranchCoverageForMediumComplexity  = 80
  static def minBranchCoverageForLowComplexity     =  0
  static def minBranchCoverageForTrivialComplexity =  0

  static def toleratedViolations = 18

  static def toleratedExceptions = [
          "de.kontext_e.tim.jsf_ui.apps.LogExportServlet.readLogfileFromPath", // some comment
          "de.kontext_e.tim.infrastructure.tools.BuildVersion", // some comment
  ]

  static def packagesWithoutTolerations = [
          "de.kontext_e.tim.entities",
          "de.kontext_e.tim.domain",
          "de.kontext_e.tim.infrastructure.queries.QueryBuilder",
  ]
}
```

Stellschrauben

- Definition der Komplexitätsbereiche
- Mindesttestabdeckung pro Komplexitätsbereich
- Anzahl erlaubter Schwellwertunterschreitungen
- saubere Pakete/Namensräumen


CCD
Unit Tests
Kleinkram
Zyklen
Statische
Codeanalyse

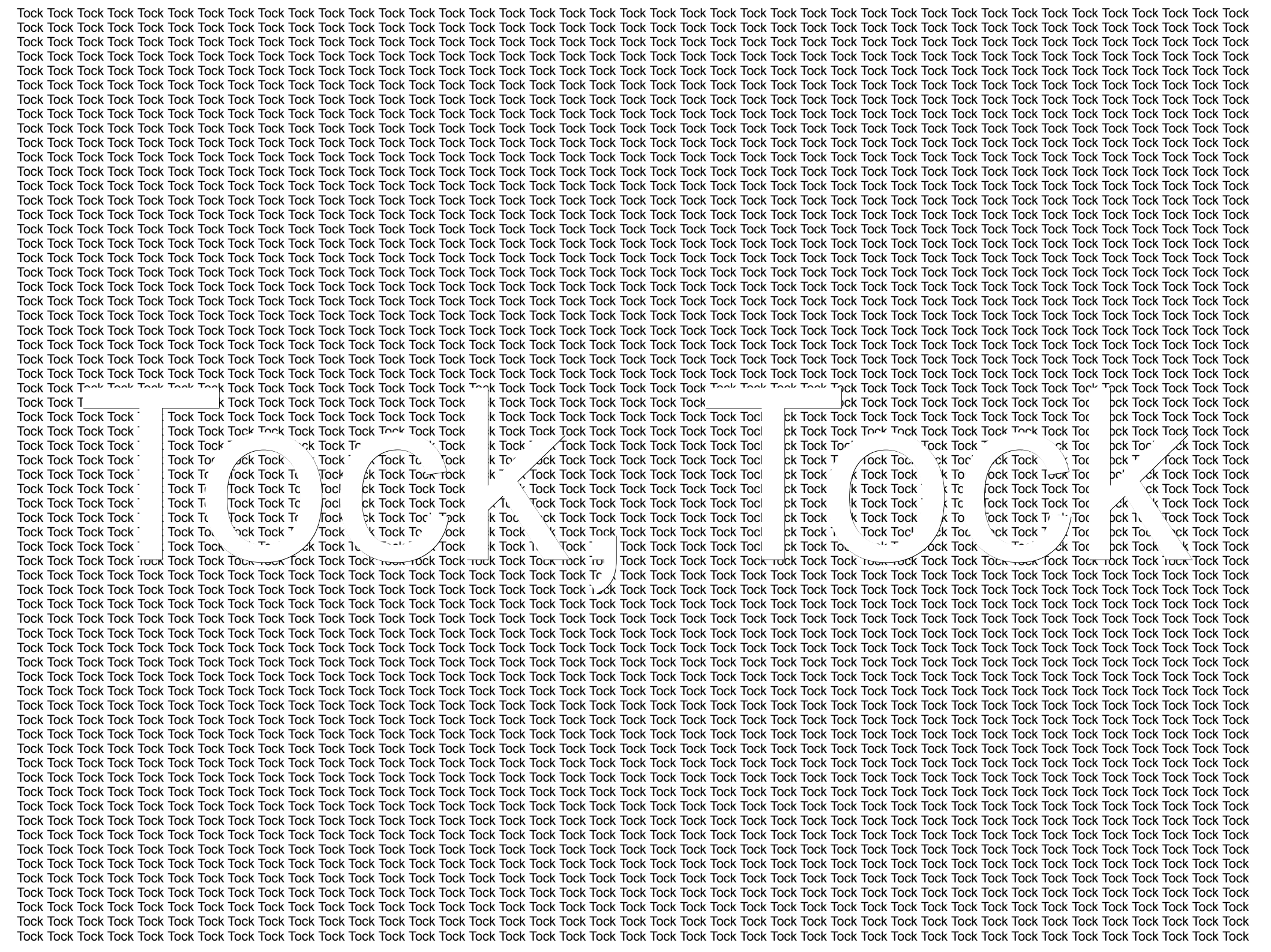
Brownfield!

Metriken?

Aaaaaaaarrrrrghhhh!!

(Was nun?)

Schranken definieren CI-Server

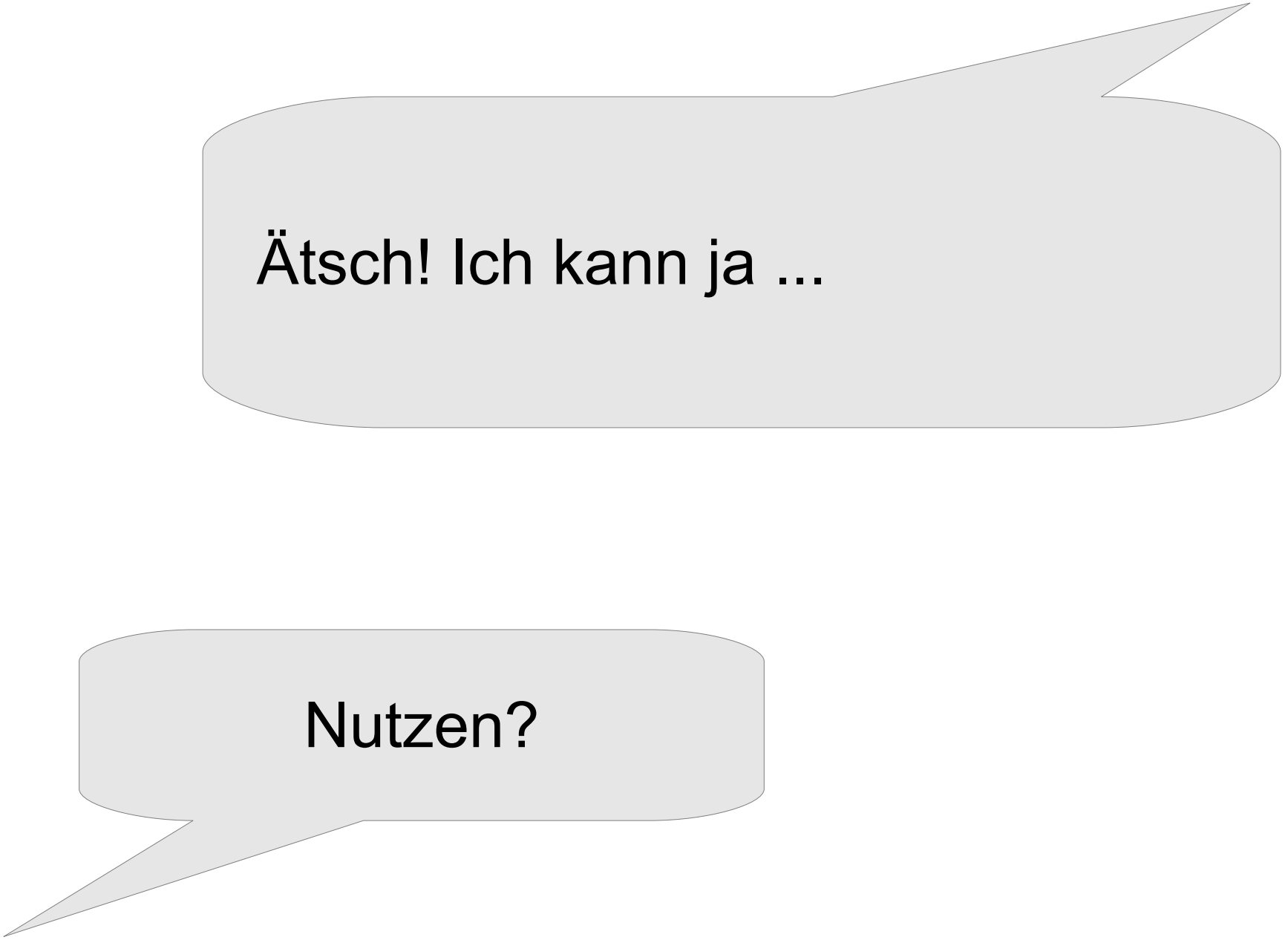

Tock

Gähn! Frust! Nutzen?

Ha! Nur für geänderte Dateien gilt...

Refactorings?

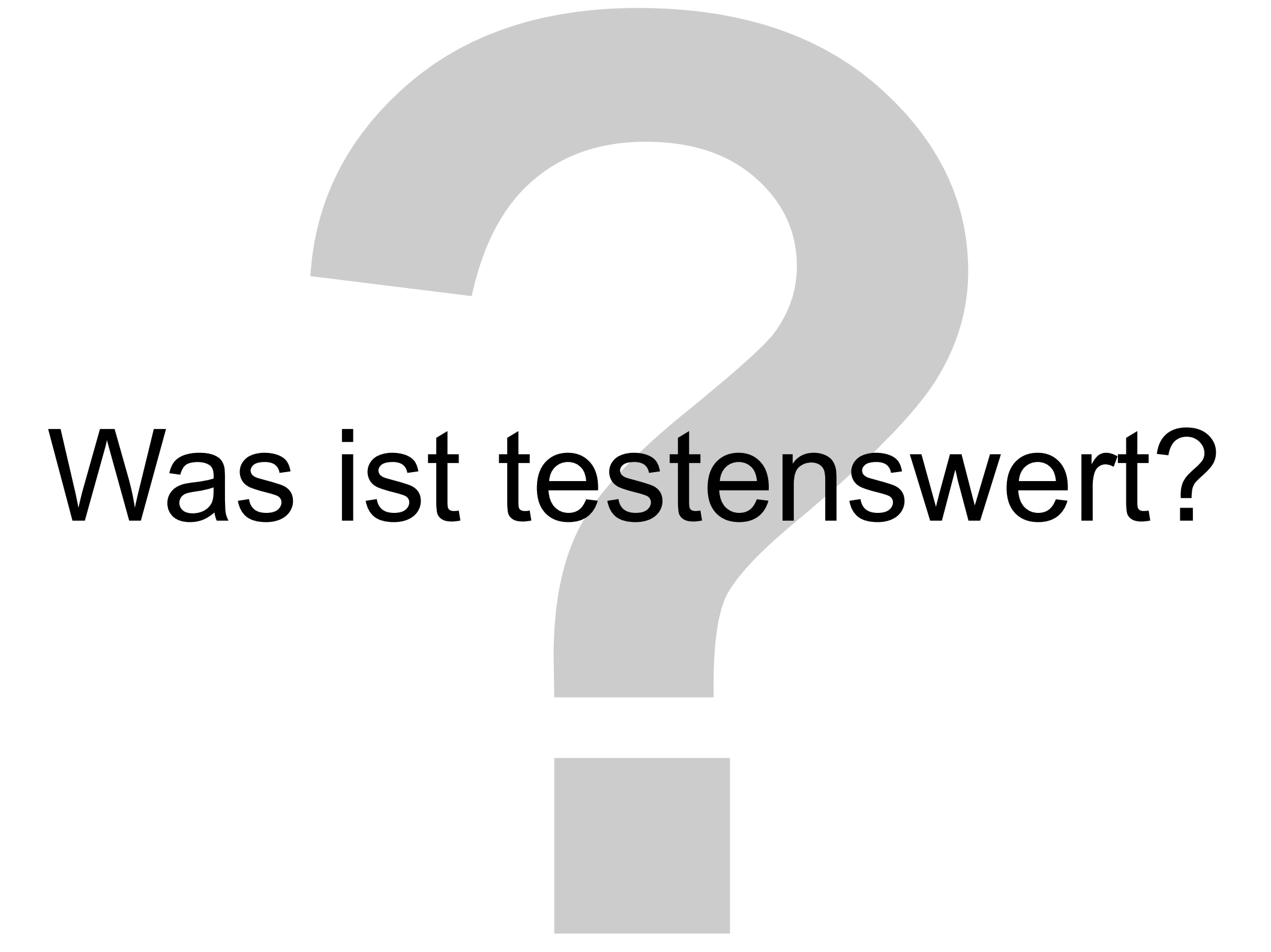
- Rename
- Remove Unused Parameter
- Introduce Parameter
- Move Class

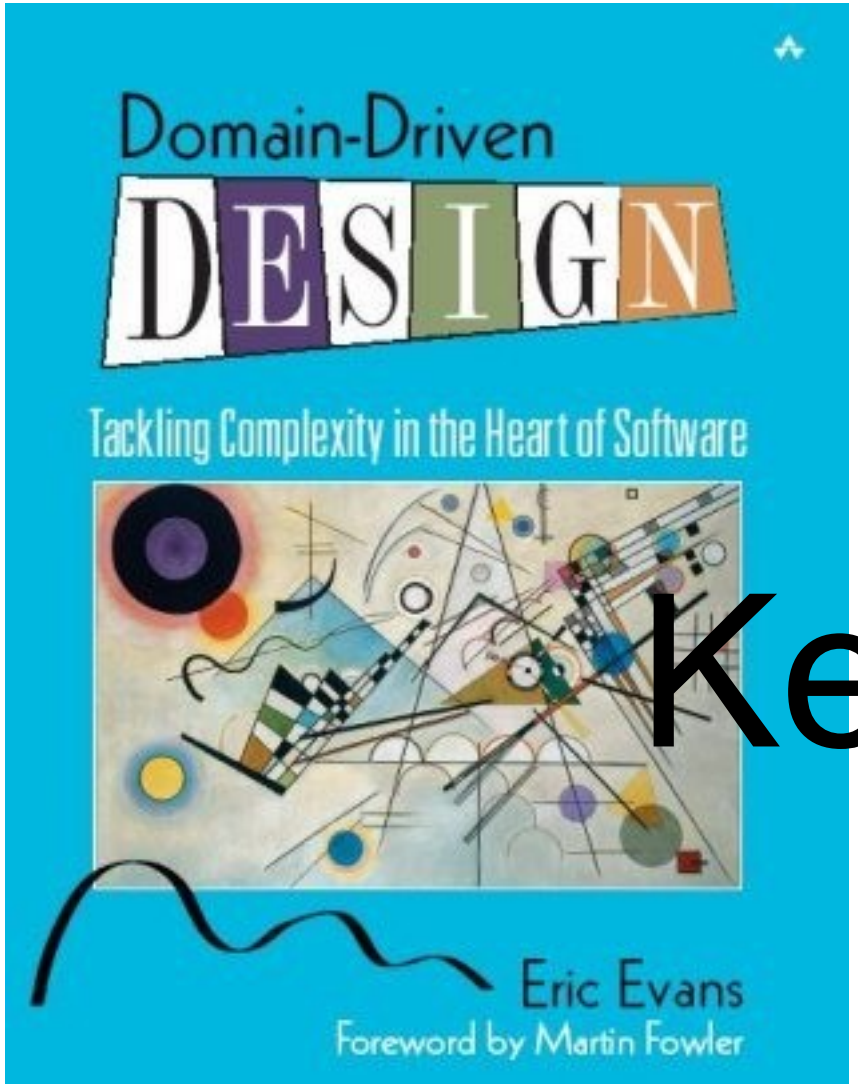

Ätsch! Ich kann ja ...
Nutzen?

Dann eben doch manuell prüfen!

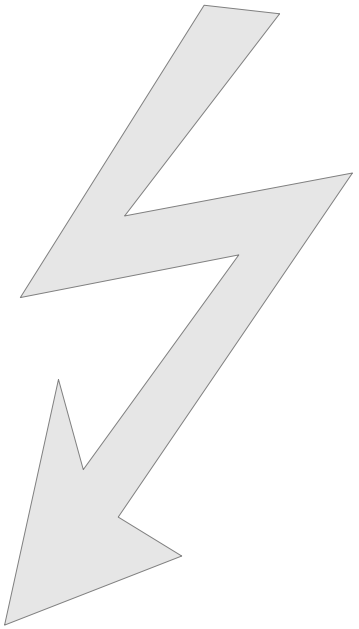
Das. Geht. Nicht. Lange. Gut.

Also wie jetzt?


Was ist testenswert?


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

Kerndomäne



Risiko:

häufige Änderungen

VCS

komplexer Code

McCabe

1. Metriken erheben (Datei)
2. Auswertung selbst übernehmen
3. Bereiche, Schranken, etc.

Stellschrauben

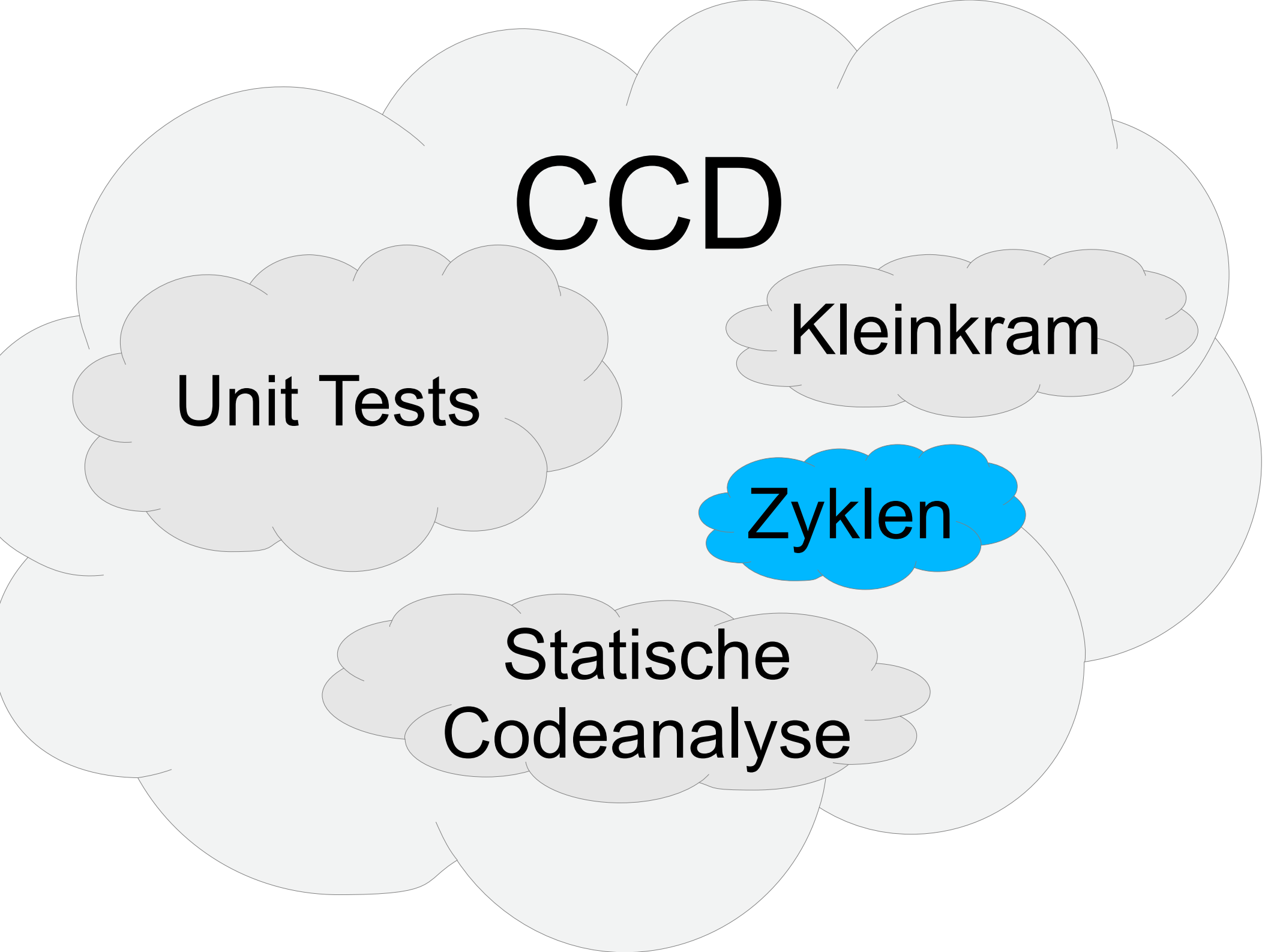
- Definition der Schwellwerte
- Anzahl erlaubter Schwellwertüberschreitungen
- Aufzählung von Ausnahmen (hash und equals von IDE)
- saubere Pakete/Namensräumen

Beispiele

- maximale Zyklomatische Komplexität
- maximale Anzahl Klassen pro Paket/Namensraum
- maximale Anzahl Methoden pro Klasse

Praxisbeispiel TiM

- JavaNCSS
- Checkstyle
- FindBugs


CCD
Unit Tests
Kleinkram
Zyklen
Statische
Codeanalyse

Kommt jetzt wieder die Nummer mit dem „Brownfield Aaaaaaaaaaargggggghhhh! Gähn! Frust! Nutzen? Was ist prüfenswert?“

Ja. Klar soweit?

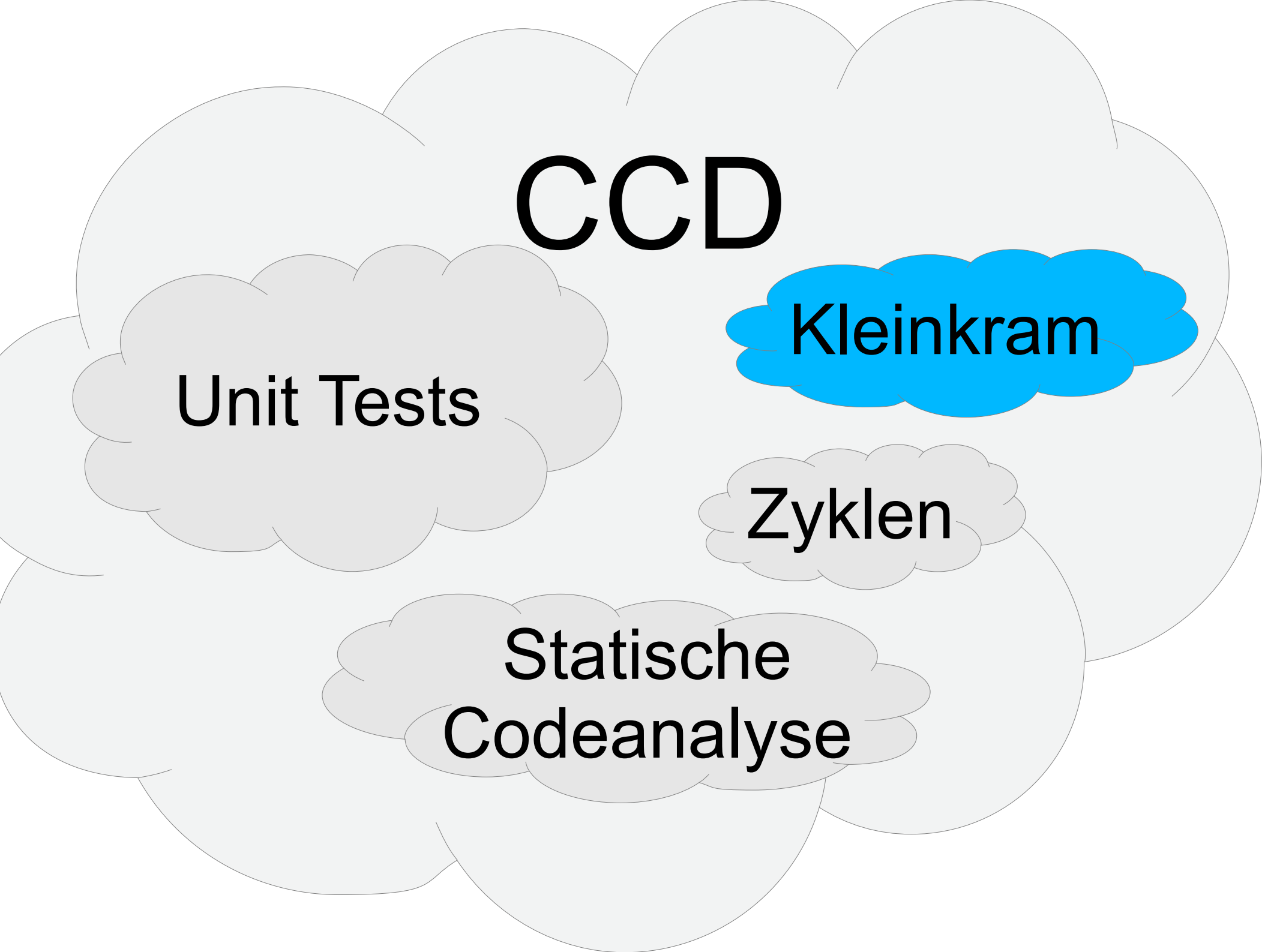
Es gibt doch Tools zur Definition und Überwachung der Architektur!

KISS: Reflektive Unit Tests

extrem flexibel
einfach, schnell, kostenlos
erlaubte/verbotene Abhängigkeiten definieren
Ausnahmen
saubere Bereiche

Praxisbeispiel

```
@Test
public void testCorrectDependencies() throws Exception {
    gatherFilesIn(startDir, sourceFiles);
    filterOutExceptionalClasses(sourceFiles, exceptionalClasses);

    assertThatNothing(dependsOn("rest_interface", sourceFiles));
    assertThatPackages(packages("rest_interface"), dependsOn("domain_model", sourceFiles));
    assertThatPackageDependsOnNothing("lowlevel_services", sourceFiles);
    assertThatPackageDependsOnNothing("domain_model", sourceFiles);
    assertThatPackageDependsOnPackages("services", sourceFiles, "lowlevel_services", "domain_model");
}
```


CCD
Kleinkram
Unit Tests
Zyklen
Statische
Codeanalyse

Ich habe in meinem Code sooo viele TODOs, da finde ich die wichtigen nicht!

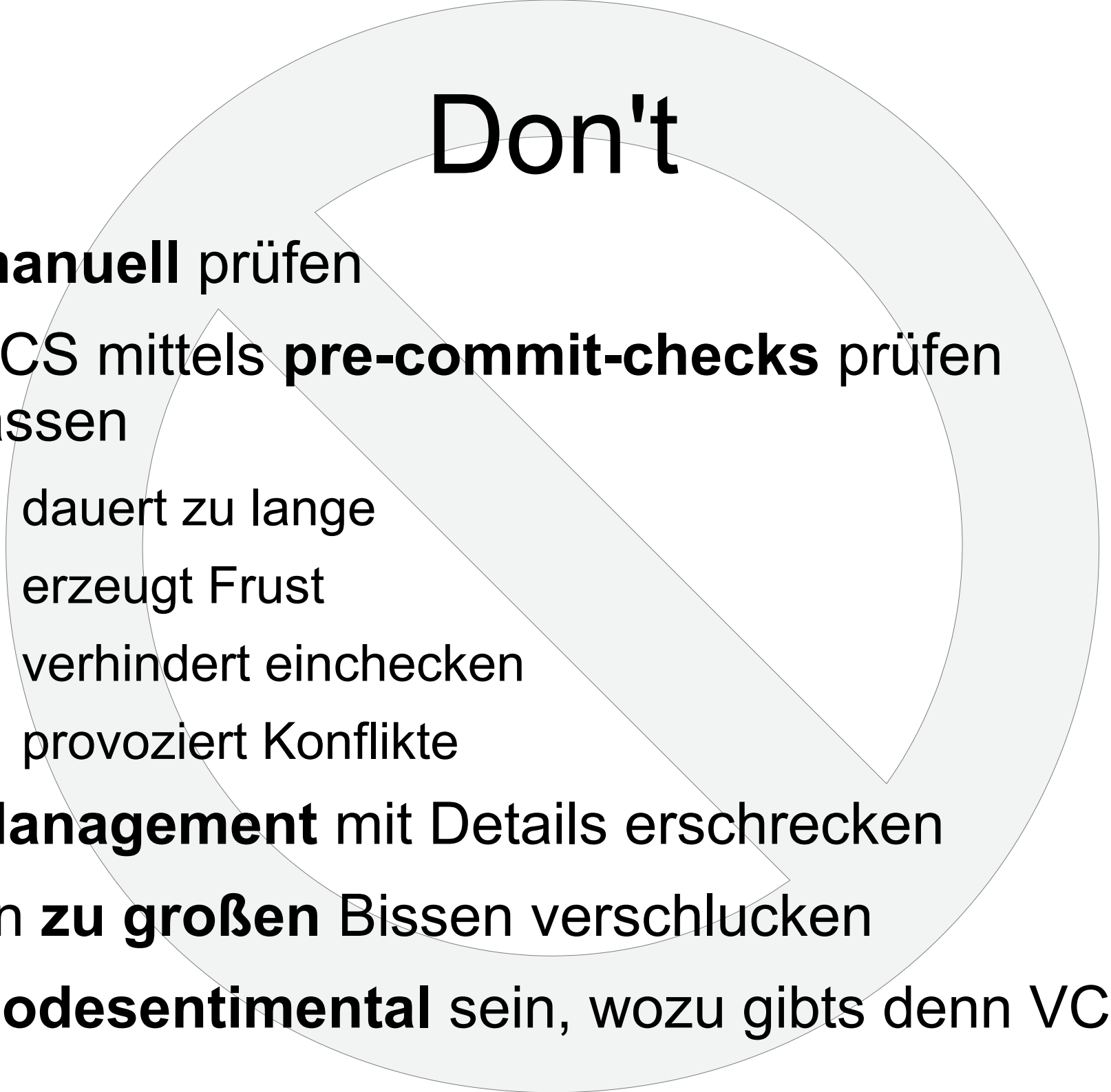
Probier **weitere „Schlüsselwörter“** mit eigener **Semantik**

TODO,
FIXME,
FIXME-BEFORE-DELIVERY,
REFACTORING,
REVIEW

Pimp my CI-Server

- hat **History**
- kann **Verlauf** darstellen und **Trends** erkennen
- kann insbesondere **Differenzen** zu **vergangenen Läufen** aufzeigen, damit man **erkennen** kann, welche der 43 Unterschreitungen der Mindesttestabdeckung neu ist

Don't



- **manuell** prüfen
- VCS mittels **pre-commit-checks** prüfen lassen
  - dauert zu lange
  - erzeugt Frust
  - verhindert einchecken
  - provoziert Konflikte
- **Management** mit Details erschrecken
- an **zu großen** Bissen verschlucken
- **Codesentimental** sein, wozu gibts denn VCS

Zusammenfassung

- Rohdaten der Metrik-Tools taugen nicht als Abbruchbedingung für CI-Build
- Nachverarbeitung
- Anzahl von Verletzungen erlauben
- saubere Bereiche definieren
- Ausnahmen definieren
- CI-Server nutzen

Zusammenfassung

- Kleine Schritte
- Stetige Verbesserung
- Team bei Laune halten
- Fortschritt muss sichtbar, fühlbar sein
- Gamification
- Code Retreat
- Legacy Code Retreat
- Kata, ... (VCS unterstützen Branches!)

Jens Nerche

Leiter Anwendungsentwicklung

Kontext E GmbH

www.kontext-e.de

Email	j.nerche@kontext-e.de
Twitter	@jensnerche
Blog	http://wp.me/p3leb2-Z

Slides	http://slideshare...
Samples	https://github.com/jensnerche
Feedback	http://speakerrate.com/jensnerche

Apropos Management

- Schätzung: nicht "Implementation" und "Refactoring" trennen, beides gehört zusammen
- Definition of Done: nicht "fertig" melden, wenn Funktion implementiert ist, sondern wenn der Code auch gut genug ist
- CI-Server rot: "seht, es läuft noch nicht"
- keine dedizierten Refactoringiterationen - constantly added value
- hochgradig personenabhängig