CONCISO.

Better Coding

Dr. Jens Bürger

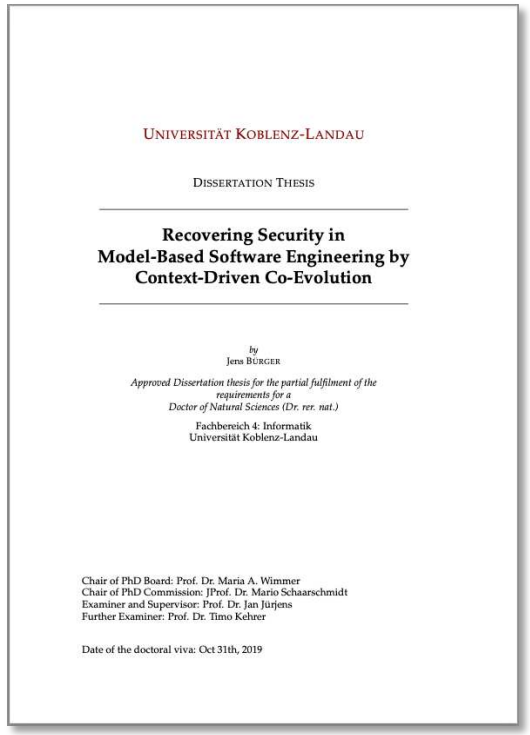
Software-Qualität und Automatisierung: Ein perfektes Paar

12.06.2024

Vorstellung

DFG

DFG Priority Programme 1593
Design For Future - Managed Software Evolution


UNIVERSITÄT KOBLENZ-LANDAU

DISSERTATION THESIS

Recovering Security in Model-Based Software Engineering by Context-Driven Co-Evolution

by
Jens BÜRGER

Approved Dissertation thesis for the partial fulfilment of the requirements for a Doctor of Natural Sciences (Dr. rer. nat.)

Fachbereich 4: Informatik
Universität Koblenz-Landau

Chair of PhD Board: Prof. Dr. Maria A. Wimmer
Chair of PhD Commission: JProf. Dr. Mario Schaarschmidt
Examiner and Supervisor: Prof. Dr. Jan Jürjens
Further Examiner: Prof. Dr. Timo Kehrer

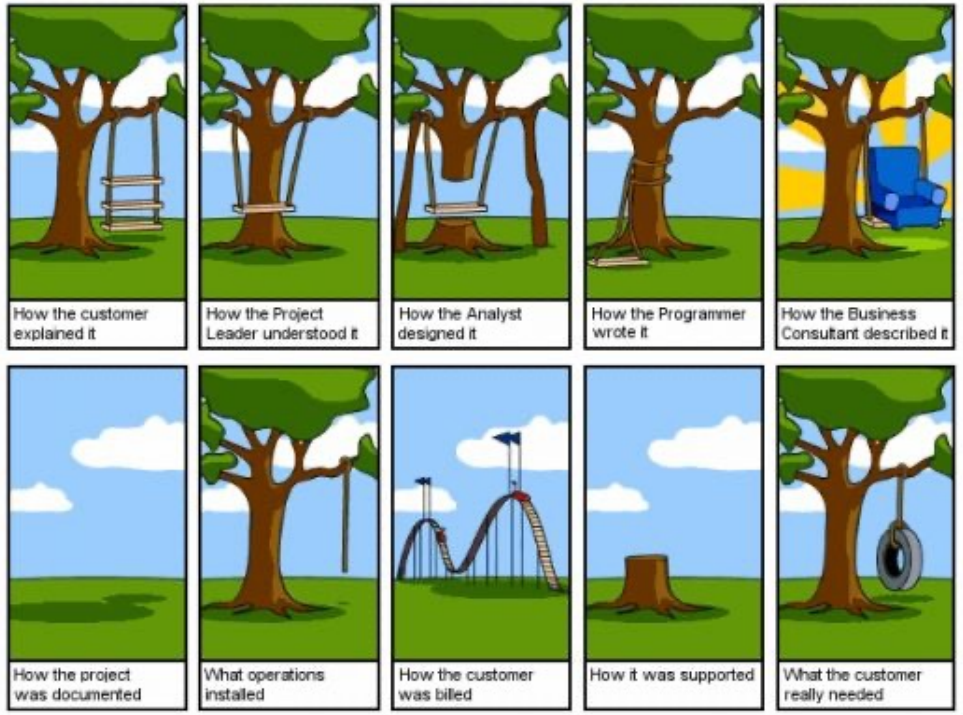
Date of the doctoral viva: Oct 31th, 2019

- Seit 2022 bei Conciso
- (Backend-)Entwicklung, Architektur
- Spezialgebiet: Technologien entfernen
- Blick über den Tellerrand

Qualität *ausbaufähig*

Motivation

- Softwareentwicklung soll Spaß machen
  → für Nutzer:innen
  - Produkt bietet Mehrwert
  - Software soll helfen und nicht behindern
  - Entwickeln nicht zum Selbstzweck
    - „Wenn du einen Scheißprozess digitalisierst, hast du einen digitalen Scheißprozess“



Motivation

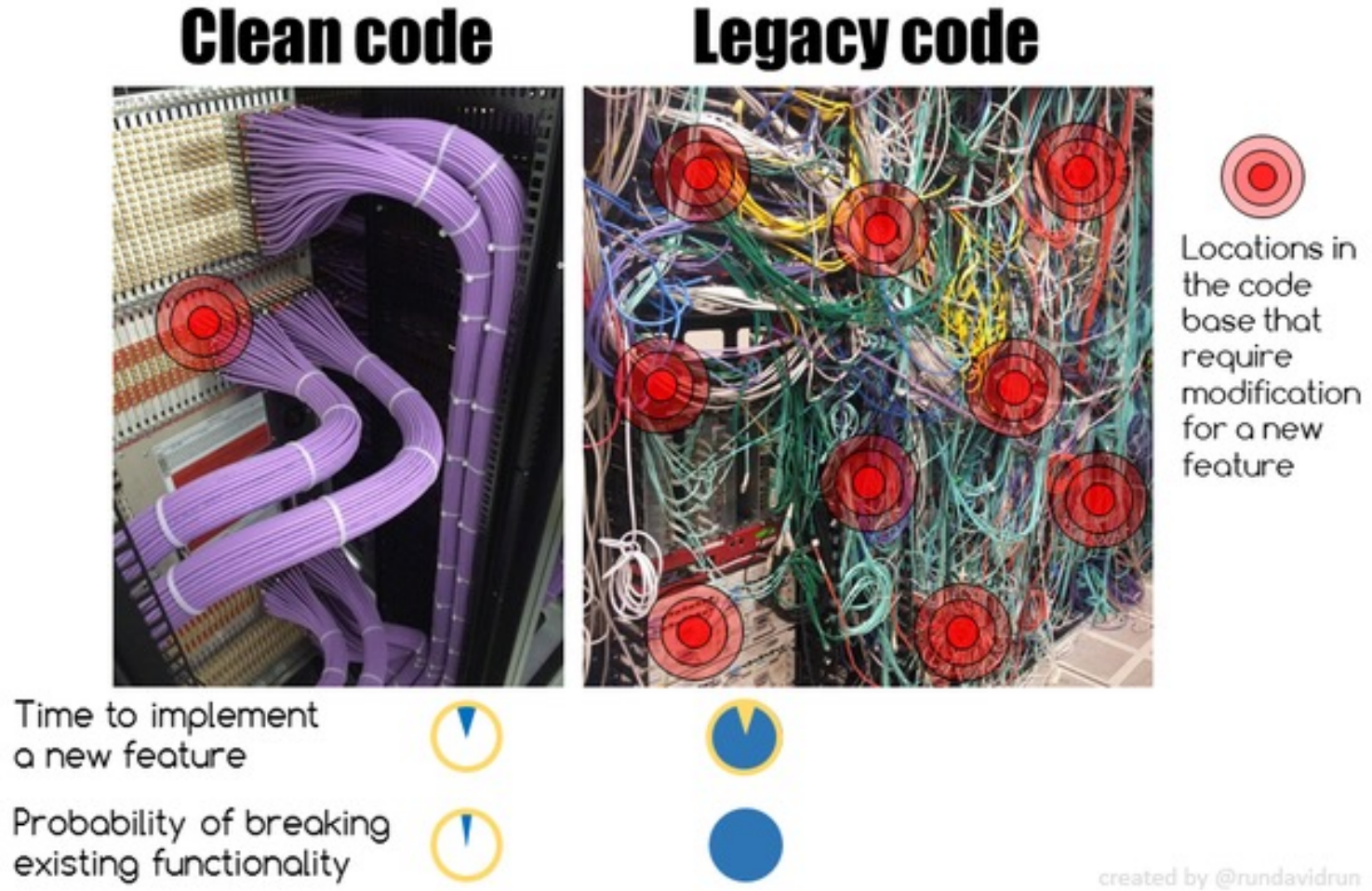
- Softwareentwicklung soll Spaß machen
  → für Entwickler:innen
  - Produkt ist verstanden
  - Plan™ existiert (und wird verfolgt)
  - Technische Schulden sind beherrscht
  - Zweckmäßige Dokumentation existiert
  - Offen für Entwicklungen außerhalb des Projekts
  - Arbeit der Teammitglieder ist geprägt von
    - Sorgfalt
    - Zusammenarbeit
    - Qualitätsbewusstsein
    - Blick für das Gesamt-Projekt und Gesamt-Team

Motivation

- Softwareentwicklung soll Spaß machen
  → für Entscheider:innen
  - Projekt ist gut wartbar
    - Analysezeit bei Bugs geringer
    - Neue Features zielgerichteter umsetzbar
  - Raum für Neues ist vorhanden
    - Sonst wird im Projekt „ausprobiert“
    - Oder es wird gar nichts mehr ausprobiert

Quelle: http://apepm.co.uk/stakeholders-functional-management/

Risiken...



Quelle: https://twitter.com/rundavidrun/status/690651629494775808

… und Chancen

DEFINITION

Rationale

(Alias: explanation, justification)

An explanation of the reasoning behind a decision, statement of requirement, design approach etc.

Rationales have the following objectives:

- **Communication.** Rationales communicate the reasoning behind various requirements or development approaches to development teams
- **Make reasoning visible.** If reasoning is made visible, defects in reasoning may be identified
- **Prevent loss of essential capabilities.** The existence of rationales often prevents essential requirements/design components from being negotiated out after a passage of time when the reasons for there inclusion have been forgotten
- **Document dependencies.** Rationales may be used to document the dependence of one capability on the existence of another thus preventing the introduction of inconsistencies by elimination or inappropriate modification
- **Prevent unnecessary rework.** If the rationale behind a complex decision is not recorded it is often lost with the passage of time. In this case development teams often doubt the validity of the decision and set about repeating the reasoning process only to come to the same result
- **Support traceability.** Rationales may be used to document the prior existing specification, design description, policy or procedure which triggered the need for a system component or capability.

Quelle: https://www.chambers.com.au/glossary/rationale.php

„Lose“ TODOs im Code

- TODOs dürfen vorkommen
- Je weniger Kontextinformationen, desto schlechter interpretierbar
- Entscheider lesen keinen Code
- → Unsichtbare (technische) Schulden
- Oft mehrere Stellen betroffen
  → Anker fehlt

```
public Schichtdrittel getSchichtdrittel(LocalDateTime time)
{
  int hour = time.getHour();
  // TODO
  if (hour < Constants.BEGINN_MITTAGSSCHICHT && hour >= BEGI
    return Schichtdrittel.FRUEH;
  if (hour < Constants.BEGINN_NACHTSCHICHT && hour >= BEGINN
    return Schichtdrittel.MITTAG;
  return Schichtdrittel.NACHT;
}
```

TODOs als deskriptiver Anker

- Erst Ticket, dann TODO
- Einheitliche Syntax für TODOs
  - mit Ticket-Referenz
  - → unterstützt Suche im Code

```
public Schichtdrittel getSchichtdrittel(LocalDateTime ti
{
  int hour = time.getHour();
  // TODO (PRJ-4630): Auf Nachtschicht verzichten?
  if (hour < Constants.BEGINN_MITTAGSSCHICHT && hour >= 
    return Schichtdrittel.FRUEH;
  if (hour < Constants.BEGINN_NACHTSCHICHT && hour >= BE
    return Schichtdrittel.MITTAG;
  return Schichtdrittel.NACHT;
}
```

(Code-)Dokumentation zum Selbstzweck

```
/**
 * the name
 */
private String name;
```

Hält auf

- Dokumentation ohne Sinn verschlechtert „signal-to-noise-Ratio“
- Cave! KI-generierte Kommentare

Unterstützt

- Dokumentieren, wenn es substanziell zum Verständnis beiträgt → Rationale
- Nicht das **was**, sondern das **wie** oder **warum**
- Perspektivwechsel:
  - Reviews übernehmen in Bereichen geringer Expertise

Knowhow-Silos zulassen

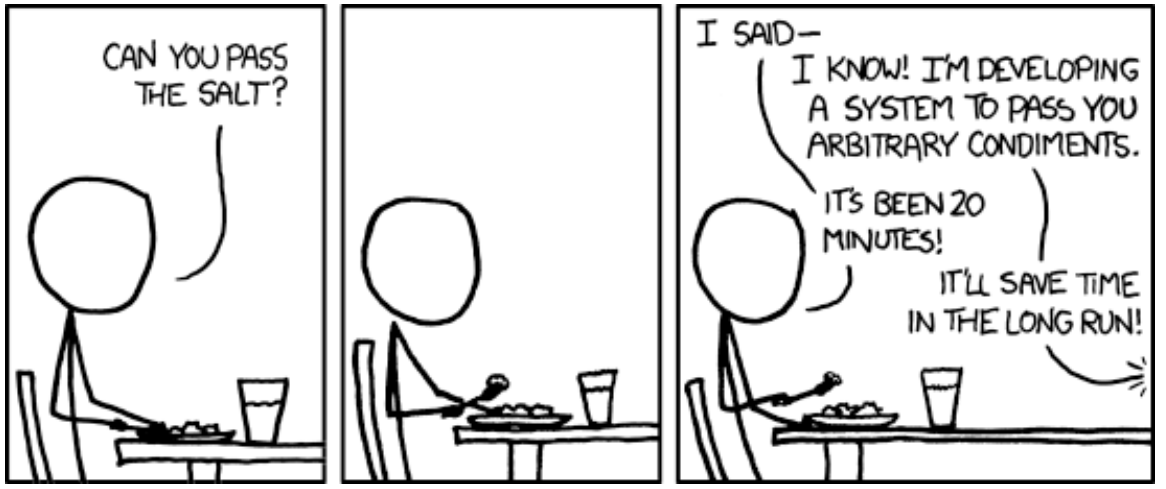
- „Klaus, du bist doch der Experte für Message Queues. Das Ticket 4711 dreht sich um Message Queues, möchtest du das dann übernehmen?“
  - → Wissensinseln
  - Kein organischer Aufbau von Expertise
  - Austausch bleibt aus
  - Reviews werden schwierig
  - → Schlüsselperson-Risko steigt

Knowhow-Silos vermeiden

- Teammitglieder explizit mit einbeziehen
- Hemmschwellen senken → Tickets kleiner schneiden
- Implementierung mit Pair Programming, Mob Programming, …

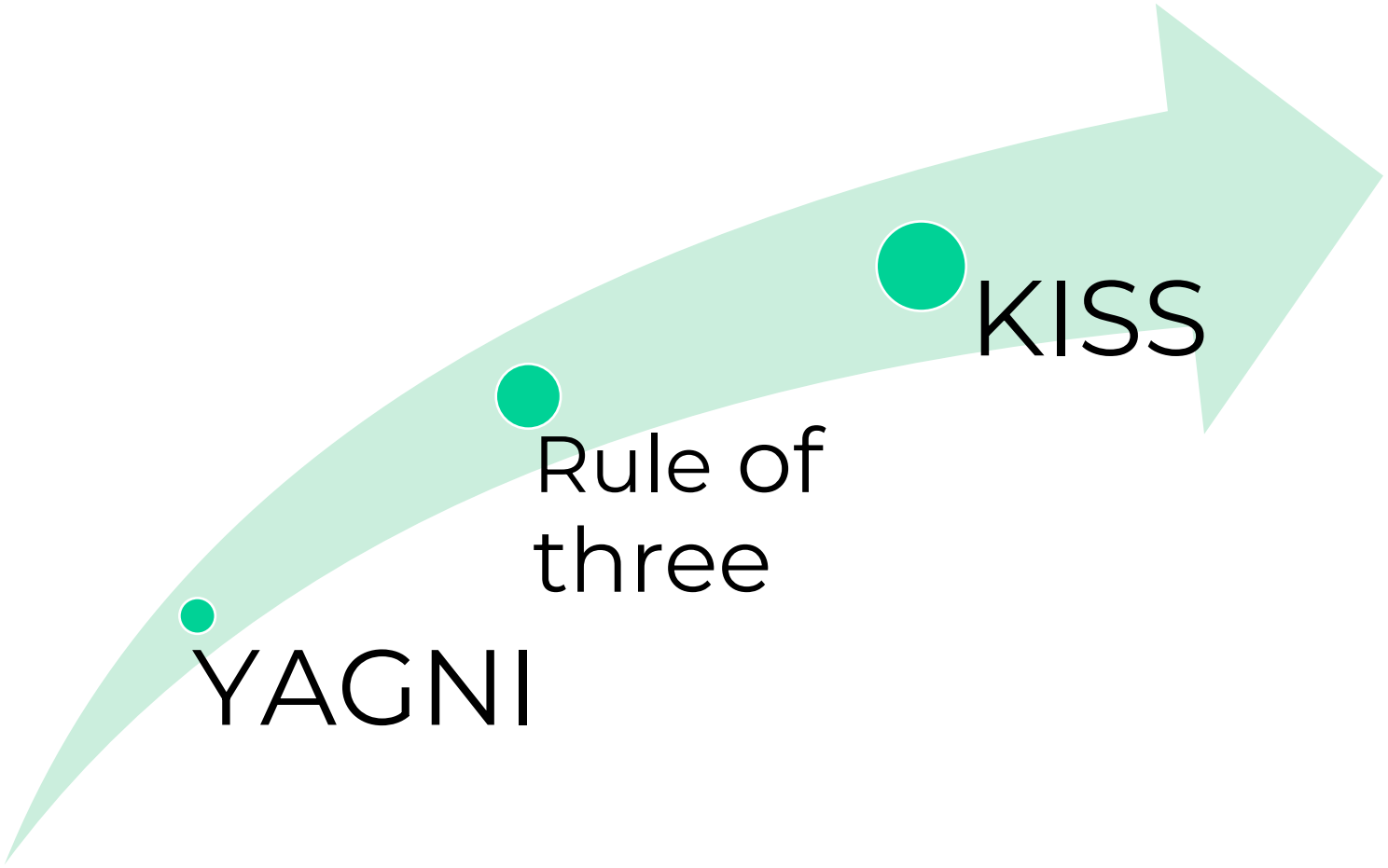
(Noch) nicht benötigte Strukturen implementieren

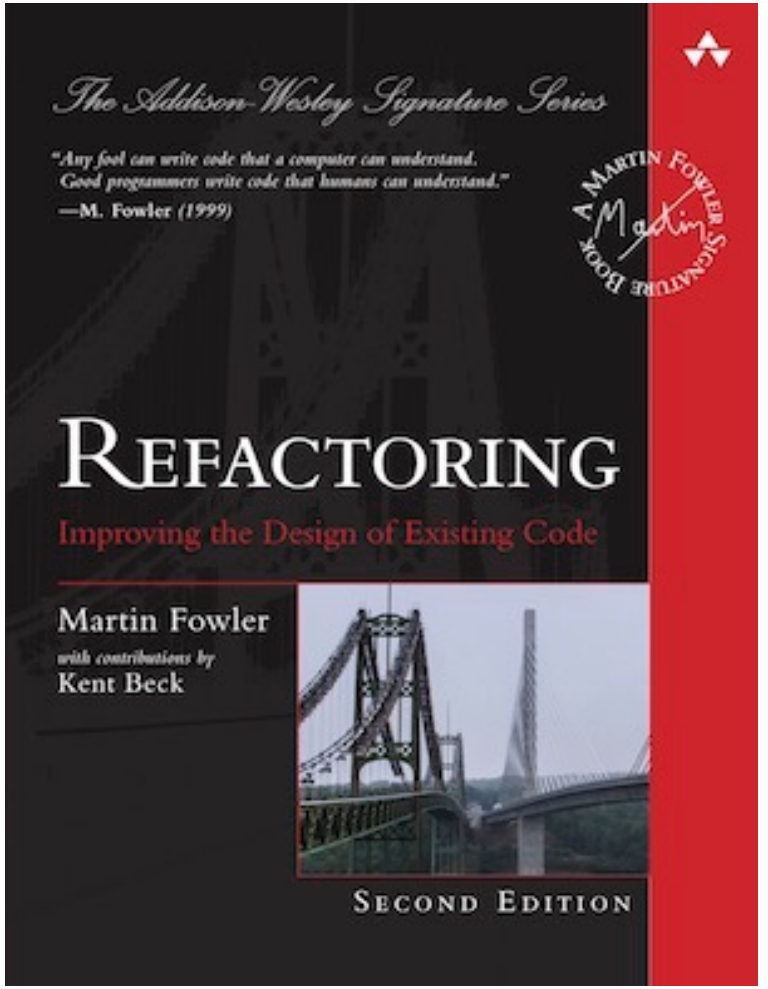
- „Aber wir wollten doch für künftige Entwicklungen vorbereitet sein!“
- Passt neues Feature nicht in Bestand → „globalgalaktische“ Lösung
- Führt zu:
  - Viel Struktur mit wenig Nutzen
  - Erschwertem Debugging
  - Erschwerter Pflege



Quelle: https://xkcd.com/974/

Nur notwendige Strukturen implementieren



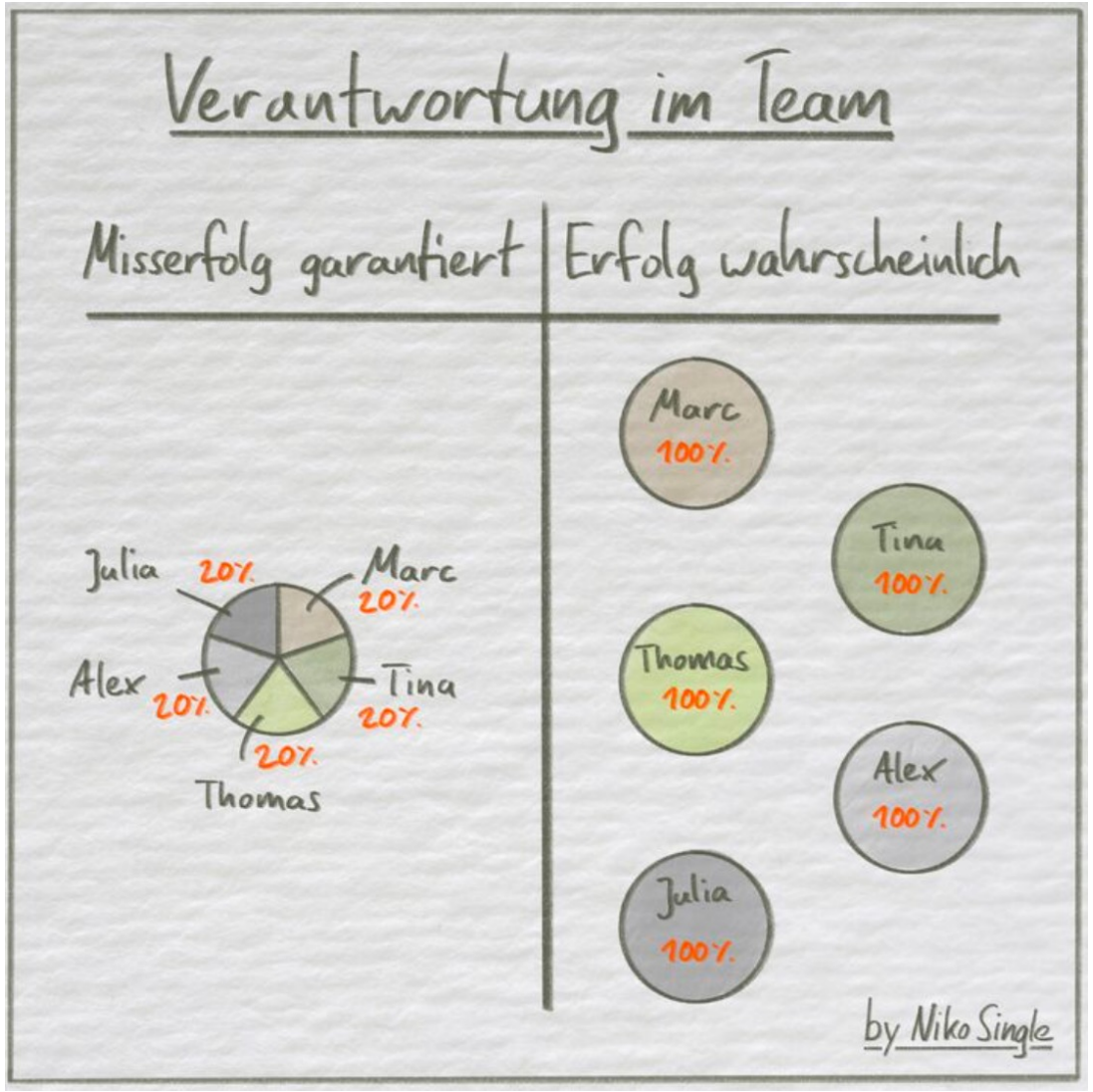


Tunnelblick-Coding

- Ausschließlich die konkrete Aufgabe umsetzen, Smells oder Optimierungspotential ignorieren
- „Das ist doch nicht **mein** Code, warum soll **ich** den ändern?“
- „Ich will jetzt das Feature implementieren und nicht aufräumen.“

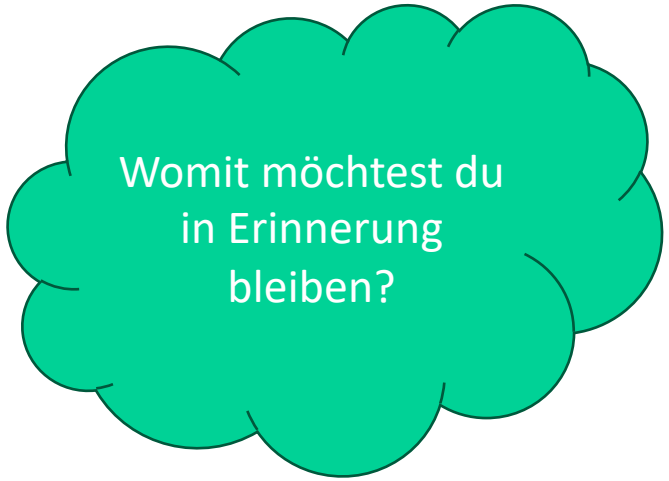
Verantwortung übernehmen

- Pfadfinder-Regel
- Collective Code Ownership
- → Erfolg wird am Produkt gemessen, nicht an Einzelleistungen



Commit-Messages für Eingeweihte

- `intermim checkin`
- `Zwischenstand`
-
- `work`
- `fix`
- `wip`
- `minor changes`
- `Add annotation`
- `Methode berechneTabelle überarbeitet`



Commit-Messages als Unterstützung

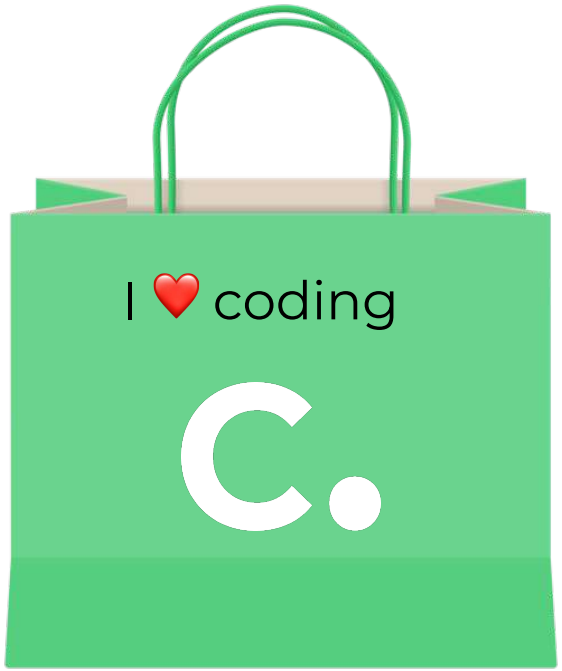
- Git-Historie zentrales Werkzeug für Recherchen im Code
  - Antwort auf die Frage „Wie kam es dazu?“
- Nicht das „Was“ dokumentieren, sondern:
  - „Wie“, „Warum“, Kontext
  - → Rationale
- Commit-Message: Feste Syntax, beginnt **immer** mit Ticket-ID
  - → Conventional Commits
- → Nach Merge sind Ticket-Zusammenhänge weiter nachvollziehbar

Goodie-Bag

☹️ Schlechter Code verschwendet Ressourcen:

- Lebenszeit
- Kreativität / Produktivität
- Geld

→ Softwareentwicklung macht keinen Spaß



🤗 Kleine Änderungen im Alltag können viel bewirken:

- Mehr Zufriedenheit → Mehr Spaß bei der Arbeit
- Wartbarkeit erhöhen → Bereit für die Zukunft
- Ressourcen sparen → Mehr Zeit für Anderes

„Qualität ist, wenn der Kunde zurückkommt, aber nicht das Produkt.“

Hermann Tietz (1837-1907),
Gründer der Kaufhauskette „Hertie“

CONCISO.

Vielen Dank für die Aufmerksamkeit!