NICHT-TRIVIALE SOFTWARE ENTWICKLUNG

Robert Kleinschmager | github.com/barclay-reg

WIESO NICHT-TRIVIAL?

Coding

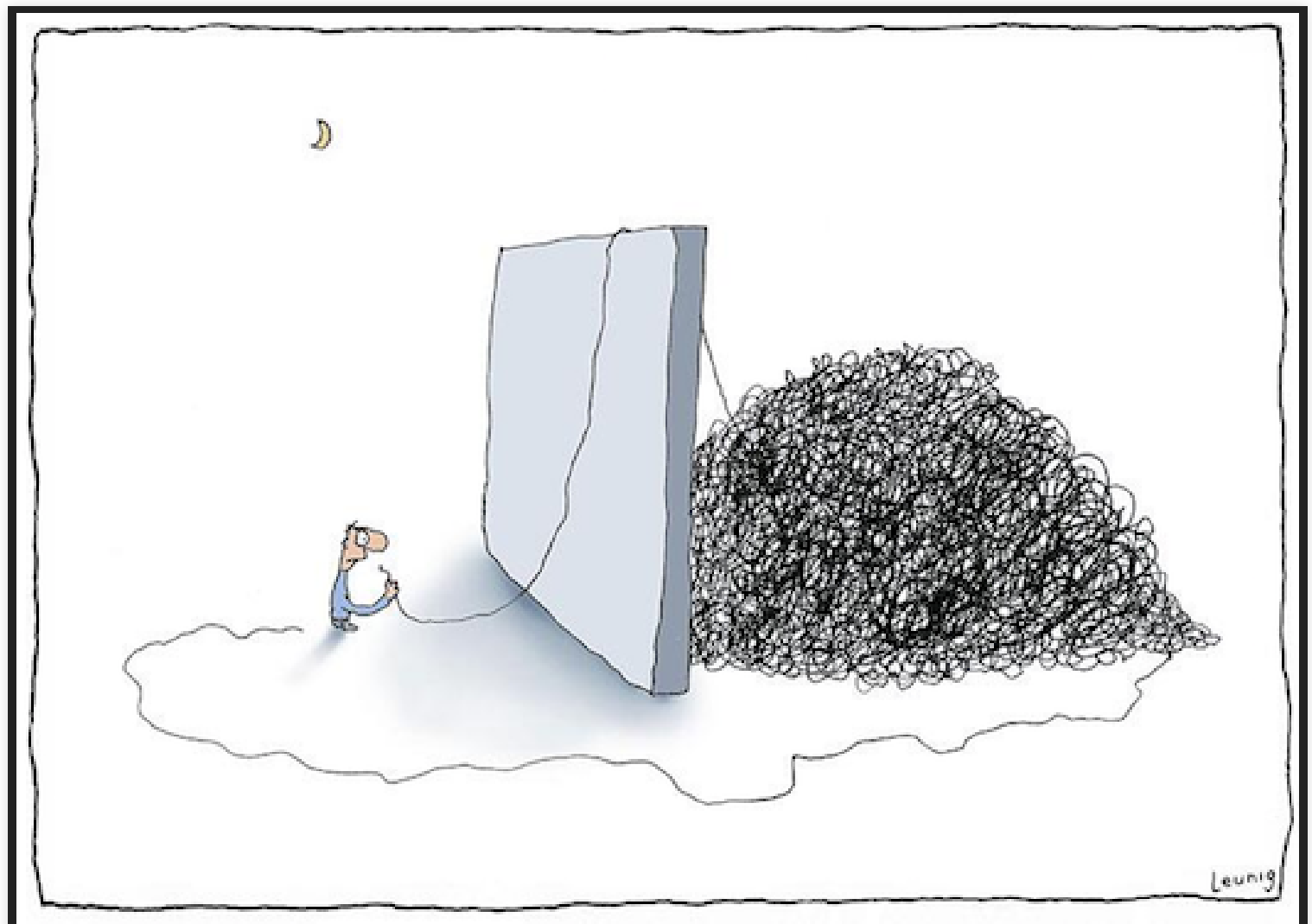
Coding | Testing

Design | Coding | Testing

Design | Coding | Testing | Documentation

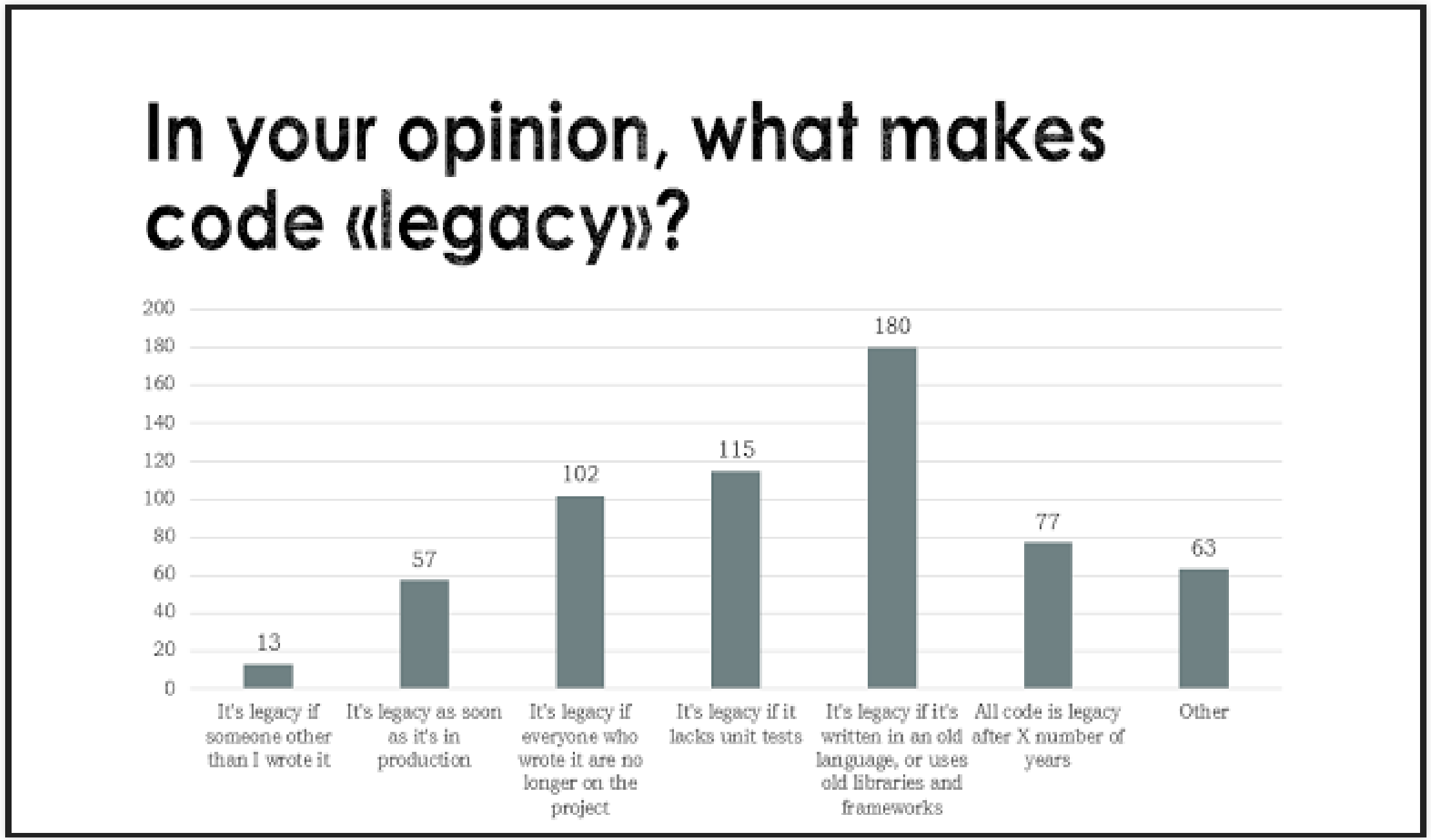
Analysis | Design | Coding | Testing | Documentation

LEGACY CODE



$\mathcal{O}(n)$ **Computer Science** @CompSciFact · Jun 13
RT @KenScambler: Legacy. Looks easy; should be done in half an hour I reckon.
pic.twitter.com/XYjlzKtT71
Reply Retweet Favorite

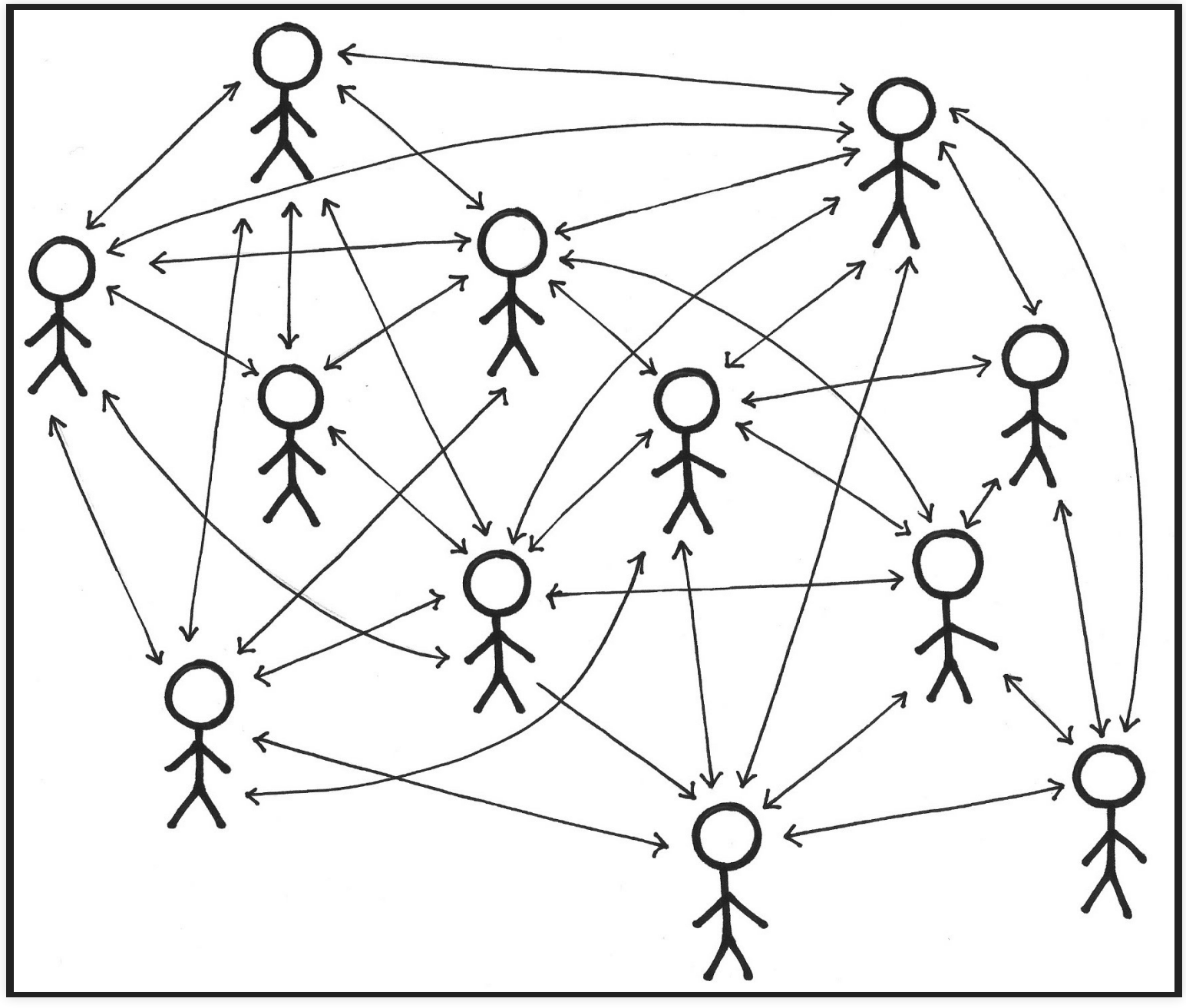
LEGACY CODE - UMFRAGE



RAHMENBEDINGUNGEN

- Funktionale Requirements
- Nicht-Funktionale Requirements
- (alte) Werkzeuge
  - IDE
  - Bibliotheken

TEAM



KONSEQUENZEN

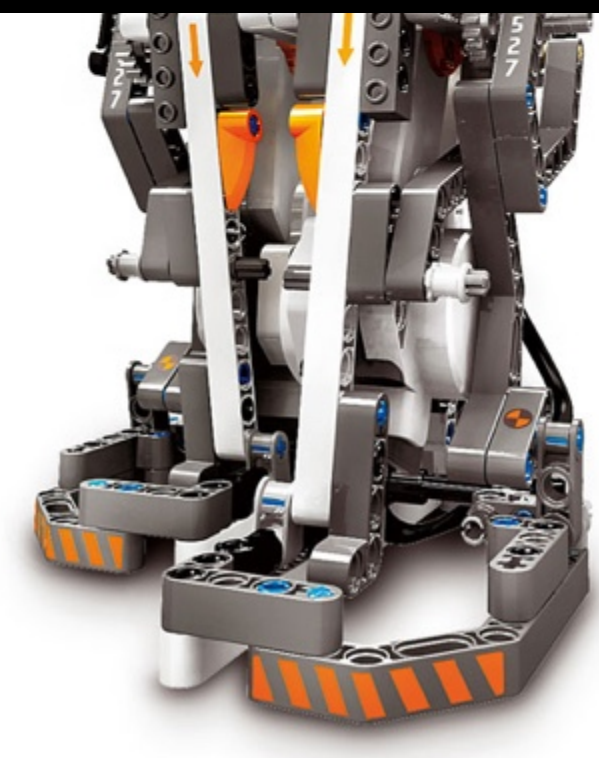
KOMPLIZIERT

- das Maß an Ungewissheit
- Wissen fehlt
- Kompliziertes System
  - zuverlässig, exakt
  - von außen steuerbar
  - konstantes Verhalten

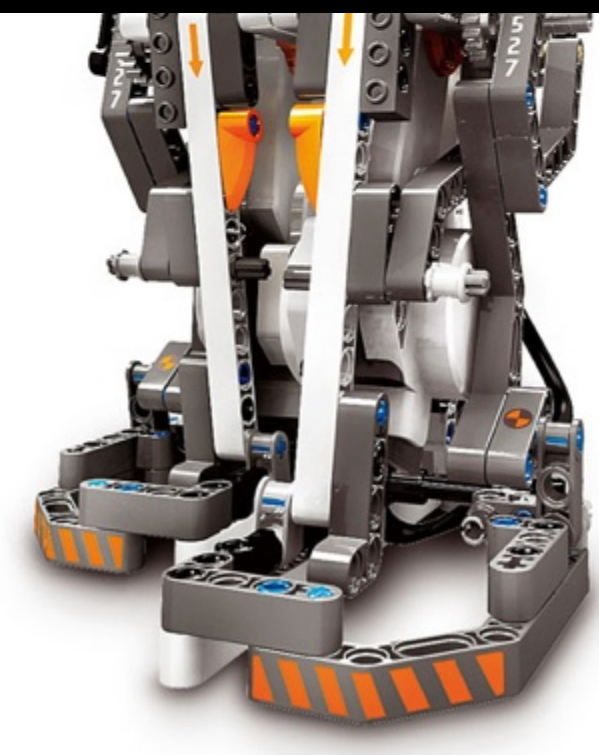
KOMPLEX

- das Maß der Menge an Überaschungen
- Kontext-abhängig
- Komplexes System
  - "lebendig", erzeugt Überaschungen
  - von außen nur beobachtbar
  - dynamisches Verhalten

- Kompliziert oder Komplex?



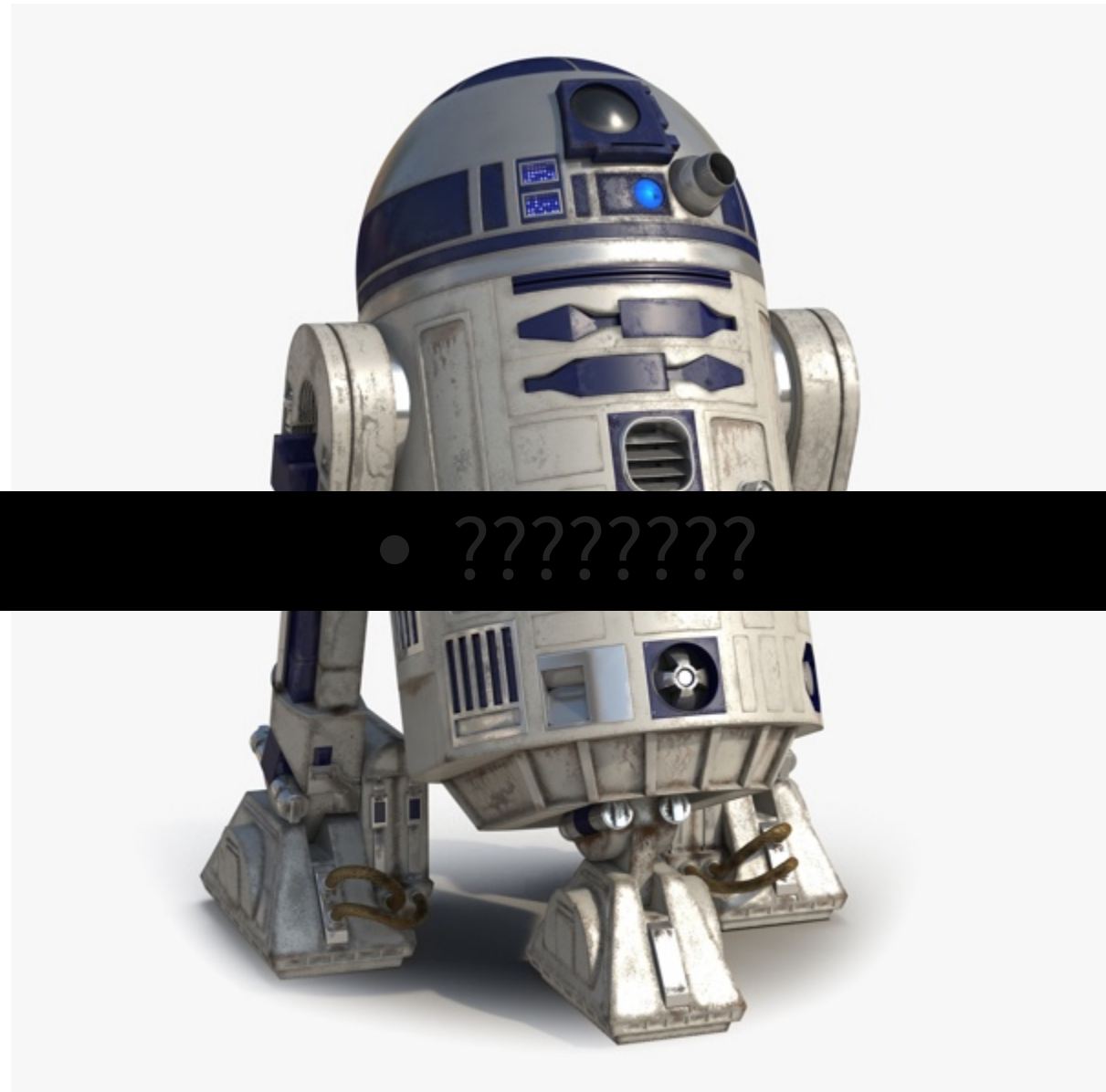
- Kompliziert



- Kompliziert oder Komplex?

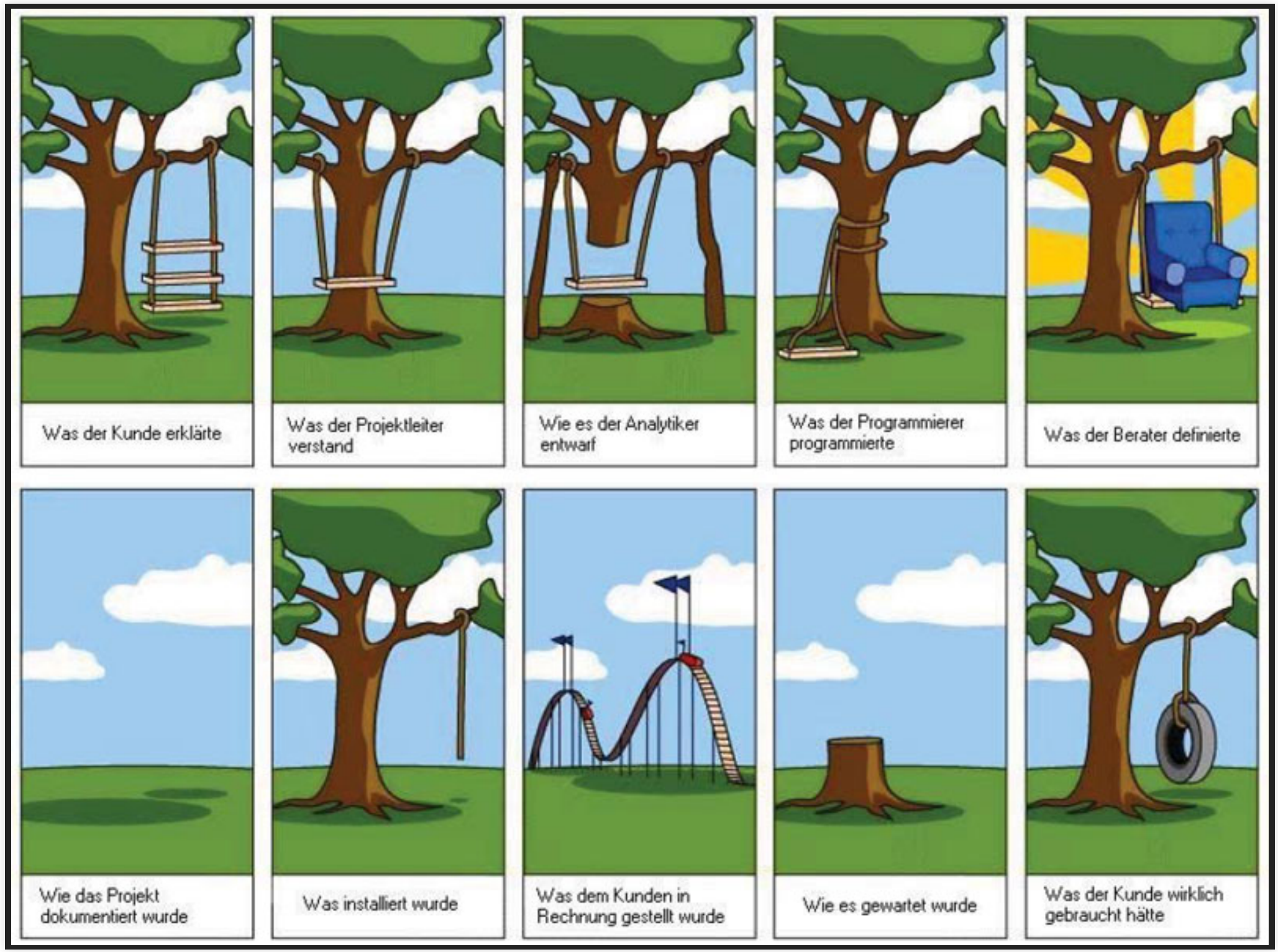
- Komplex

- Kompliziert oder Komplex?



- ????????

Software wächst in der "freien Wildbahn" in einem Spannungsfeld


Was der Kunde erklärte
Was der Projektleiter verstand
Wie es der Analytiker entwarf
Was der Programmierer programmierte
Was der Berater definierte
Wie das Projekt dokumentiert wurde
Was installiert wurde
Was dem Kunden in Rechnung gestellt wurde
Wie es gewartet wurde
Was der Kunde wirklich gebraucht hätte

- Coding wird zunehmen schwerer

ÜBERGANG VON KOMPLIZIERT ZU KOMPLEX

- Schätzungen werden ungenauer
- Analysen werden umfangreicher
- Test werden aufwändiger, vielfältiger
- Dokumentation wird umfassender
- Technische Schulden entstehen

KOMPLEXITÄT IN DER SOFTWAREENTWICKLUNG

> *software systems grow faster in size and complexity than methods to handle complexity are invented*
>
> — Wirth‘s Law

> *Programming is still a stone-age crafts*

GEGENMASSNAHMEN

CLEAN CODE

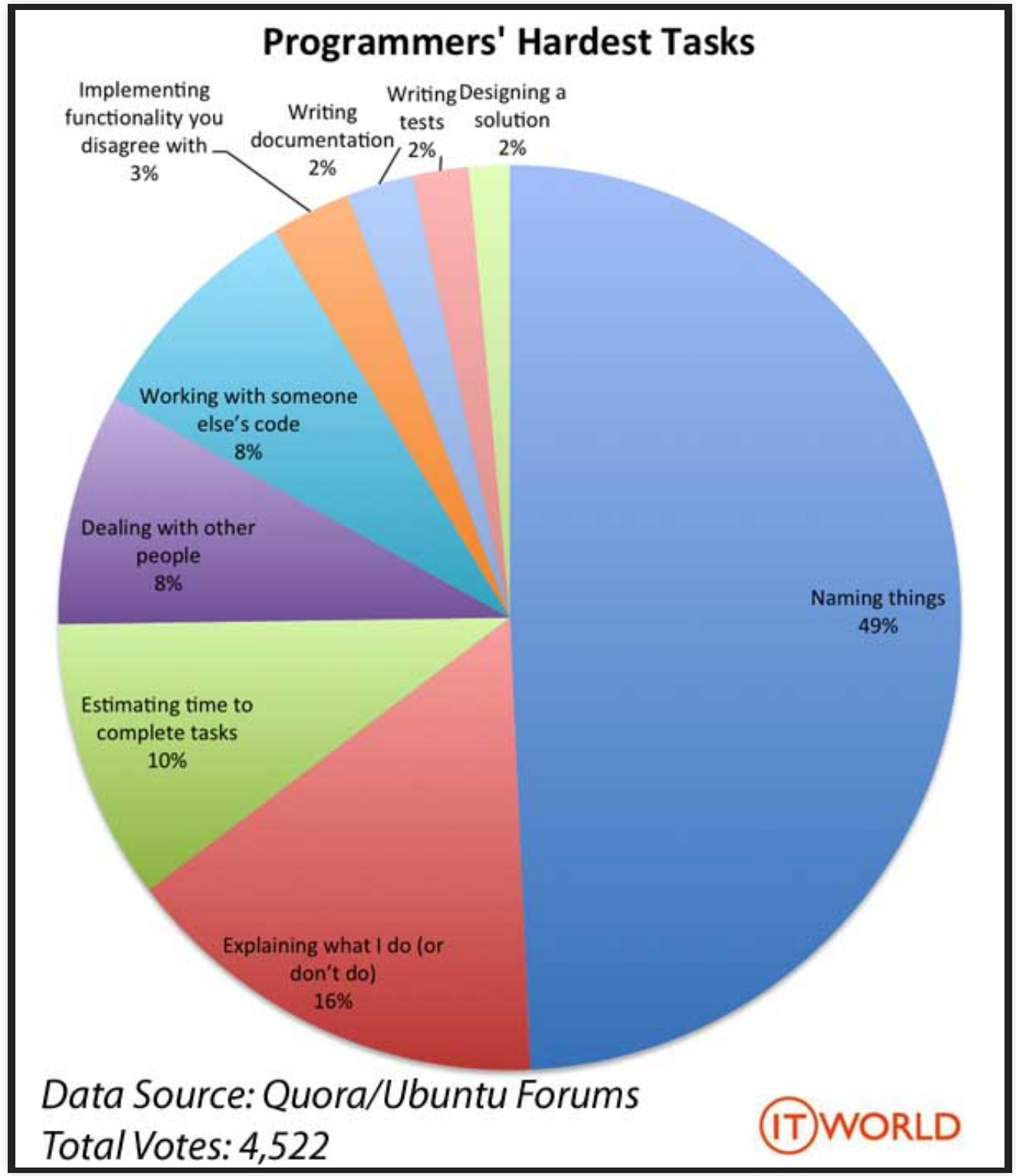
- Code wird nur einmal geschrieben, aber viele male gelesen
- Keine Code-Magneten schreiben;
  - Single Responsibility Principle (Theorie)
  - Main Purpose Principle (Praxis)
- Komposition over Inheritance

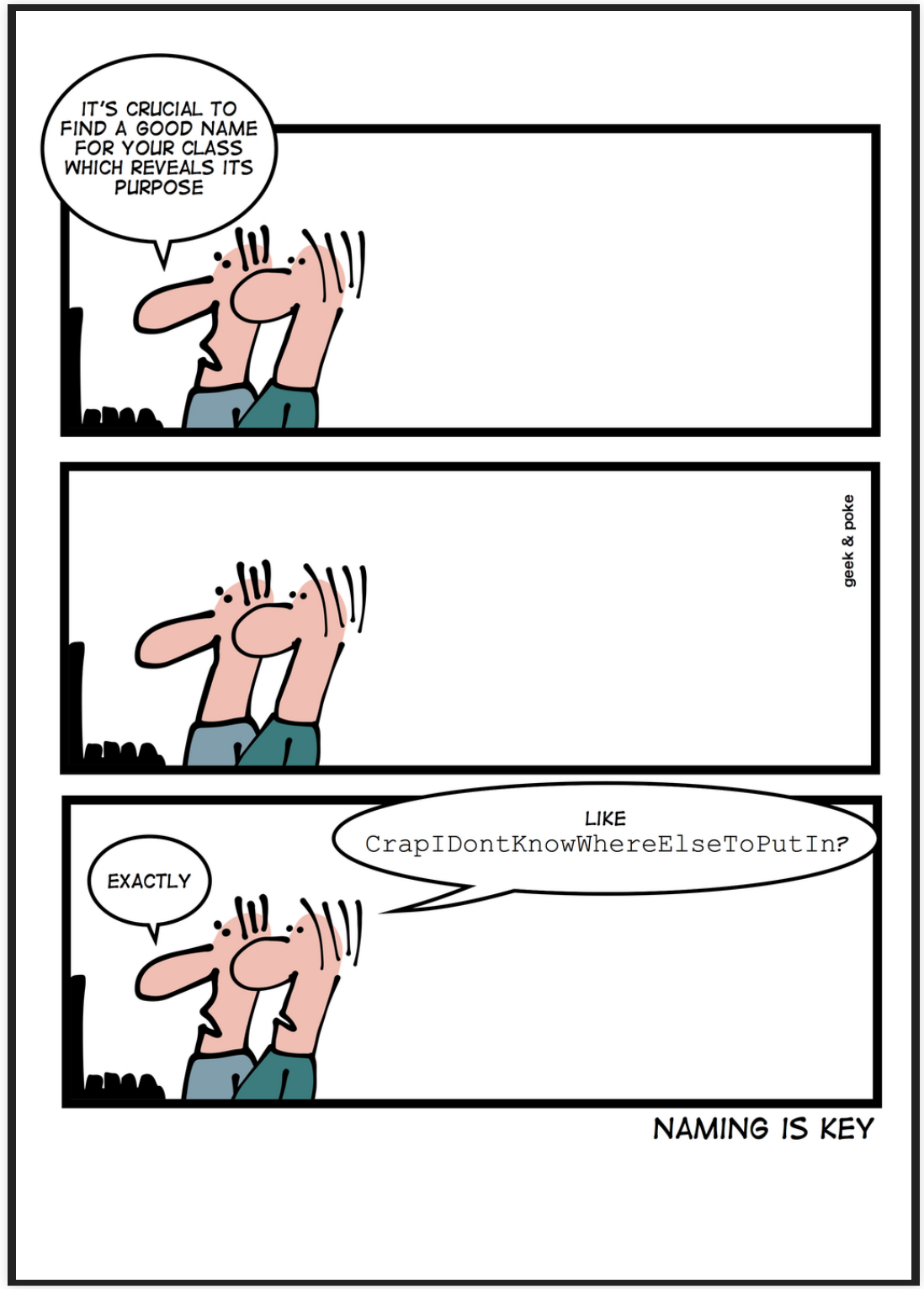
Broken Window Principle

— Robert “Uncle Bob” Martin

Leave the campground cleaner than you found it

— Robert “Uncle Bob” Martin


Programmers' Hardest Tasks
Implementing functionality you disagree with 3%
Writing documentation 2%
Writing tests 2%
Designing a solution 2%
Working with someone else's code 8%
Dealing with other people 8%
Estimating time to complete tasks 10%
Explaining what I do (or don't do) 16%
Naming things 49%
Data Source: Quora/Ubuntu Forums
Total Votes: 4,522
IT WORLD


IT'S CRUCIAL TO FIND A GOOD NAME FOR YOUR CLASS WHICH REVEALS ITS PURPOSE
geek & poke
LIKE
CrapIDontKnowWhereElseToPutIn?
EXACTLY
NAMING IS KEY

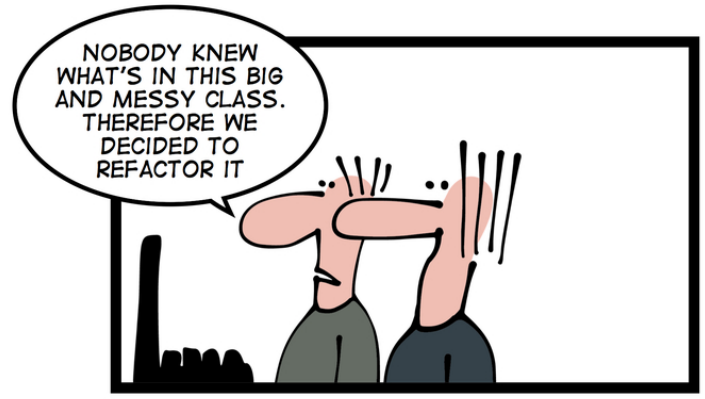
DOKUMENTATION

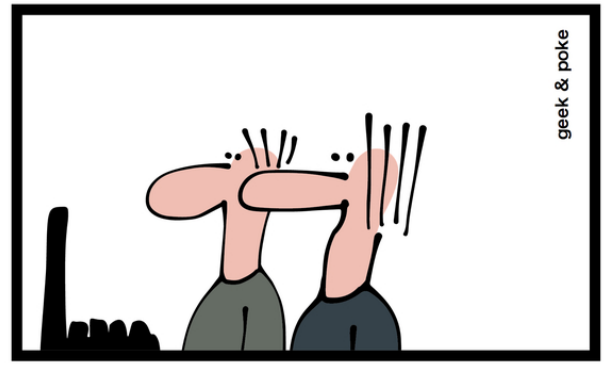
- Nur Konzepte & Ideen
  - diese möglichst nahe am Code (== gut erreichbar für den Entwickler)
- Nicht WAS, sondern WARUM
- Je expressiver der Code, desto weniger Dokumentation ist nötig

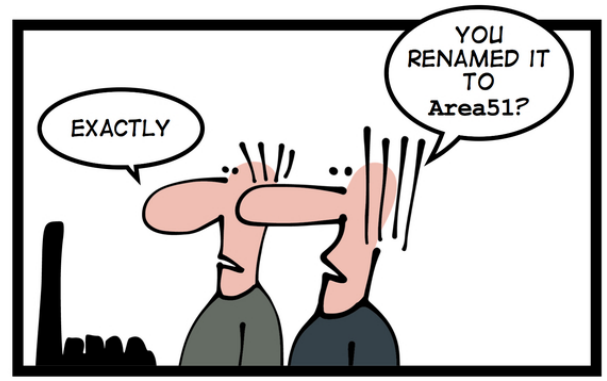
REFACTORING

Struktur verändern
ohne
Verhalten zu ändern

REFACTORING IS KEY







CONTINUOUS INTEGRATION

- Compiliere
- Paketieren
- (automatisiert) Testen
- Validieren
- Verifizieren

NACHVOLLZIEHBARKEIT

Analyse → Anforderung → Aufgabe → Code → Test → Auslieferung

METRIKEN

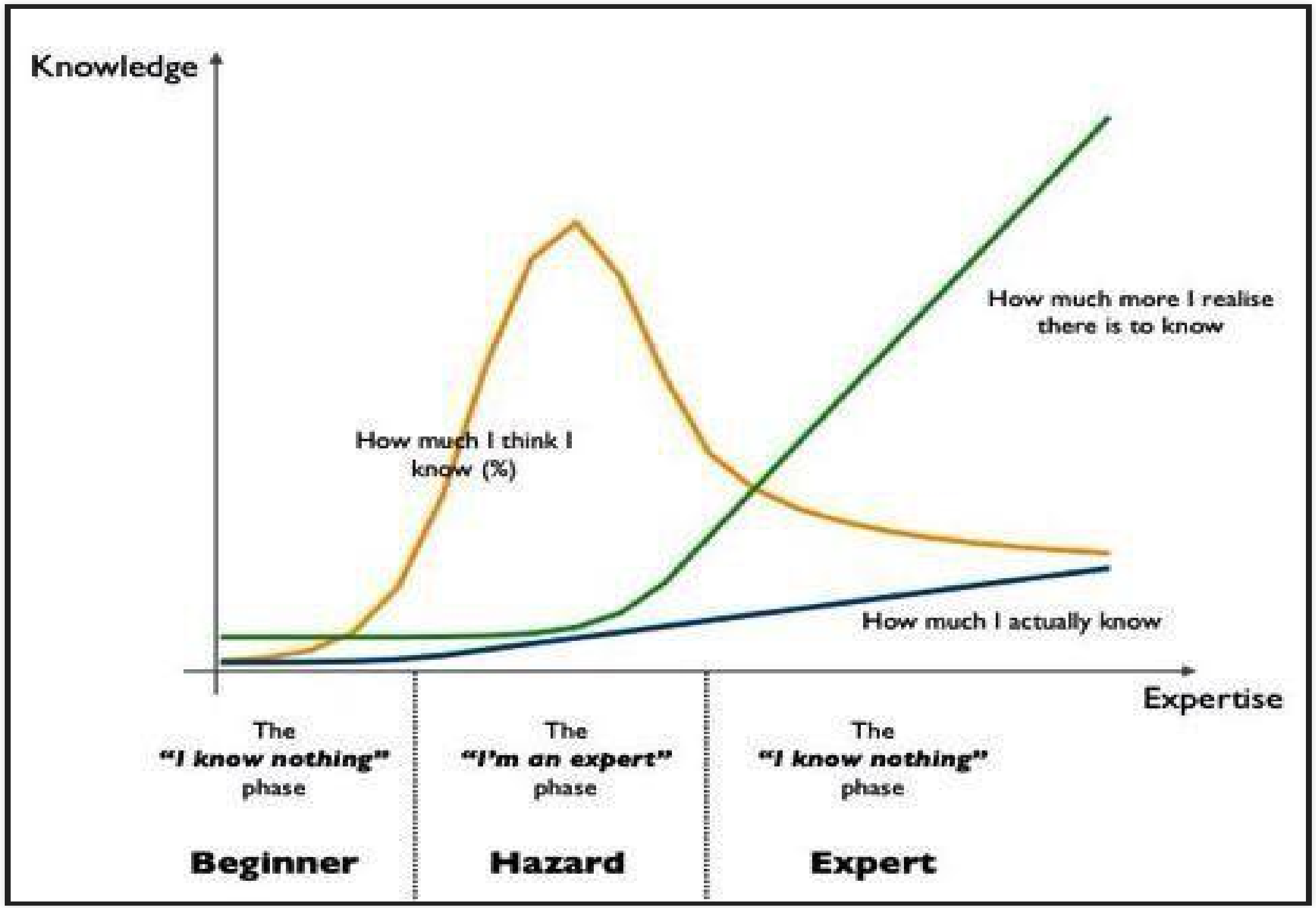
- Messen des Status-Quo der Software-Komplexität
  - Wird sie schlechter oder besser?
- Identifikation der größten Risiken
  - Wichtig für Priorisierung bei limitiertem Budget/Zeit

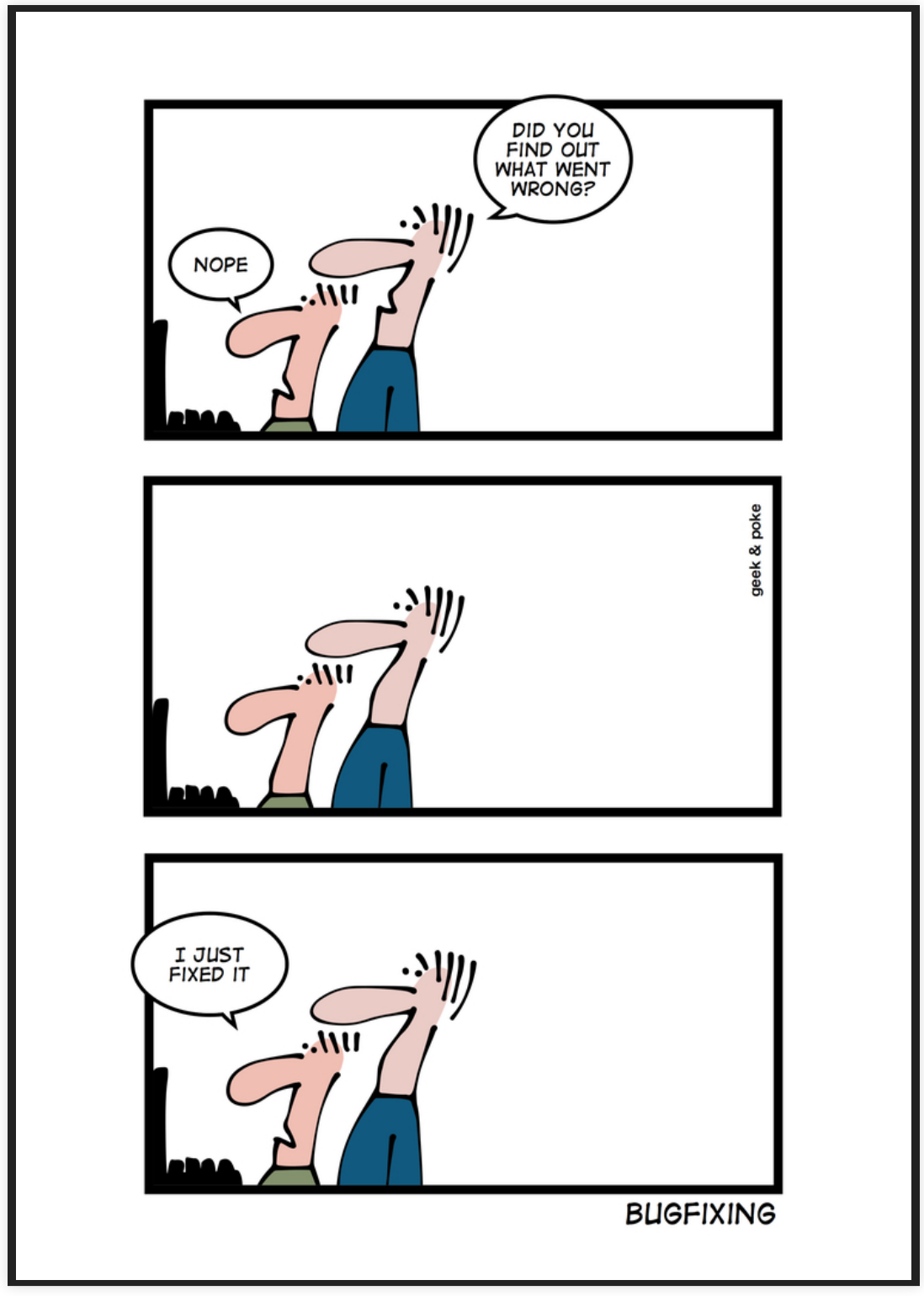
> *The only valid code metric is WTFs per minute.*
>
> — Robert “Uncle Bob” Martin

SELBSTKRITIK

- Regelmäßig den eigenen Code überprüfen

> *The worst programmer in the world is you – one year ago.*
>
> — Andy Hunt


Knowledge
How much more I realise there is to know
How much I think I know (%)
How much I actually know
Expertise
The "I know nothing" phase
The "I'm an expert" phase
The "I know nothing" phase
Beginner
Hazard
Expert


DID YOU FIND OUT WHAT WENT WRONG?
NOPE
geek & poke
I JUST FIXED IT
BUGFIXING

ANALYTISCHES TALENT

- Analyse von Kundenwüschen
  - Was will der Kunde und WARUM?
- Analyse von Fehlern/Bugs
- Debugging
- Aufstellen und Verifizieren von Hypothesen
- Bewertung von Hypothesen
  - == Erfahrung
- Schritt-für-Schritt Vorgehen
- Zyklisch verfeinern

FAZIT

Kontinuierliches **Anwenden** und **Verbessern** dieser Maßnahmen ist unsere beste *Waffe* gegen die zunehmende **Komplexität**

QUELLEN

BILDER

- Legacy Wall
  https://raw.githubusercontent.com/bettercodehub/pitch/master/assets/legacy-code.png
- Legacy Survey http://www.karolikl.com/2015/10/your-definition-of-legacy-impacts-how.html
- Team http://scrumbook.org.datasenter.no/images/SmallTeam_Pre.jpg
- Programmers hardest task
  https://www.itworld.com/article/2833265/cloud-computing/don-t-go-into-programming-if-you-don-t-have-a-good-thesaurus.html
- Naming is Key http://geek-and-poke.com/geekandpoke/2013/8/20/naming-is-key
- Refactoring is Key http://geek-and-poke.com/geekandpoke/2013/8/26/refactoring-is-key

BUCHTIPPS

The Pragmatic Programmer, From Journeyman To Master

— Andy Hunt

Clean Coder / Clean Coder

— Robert “Uncle Bob” Martin