Software Engineering 2 – Metrics. Part 2

Hochschule
für Technik
Stuttgart

Marcus Deininger

SS 2021

Overview

- Definitions
- Unit/Code Metrics
- Inter Unit/Design Metrics
- Document/Specification Metrics
- Eclipse metrics plugin
- Bonus: Industrial Software Metrics Top 10 List
- Metrics, Models and Scales
- Quality and Complexity
- Defining a Metric

A Closer Look

- How can a (quality) feature become measurable?
- How can the measured value be interpreted?
- What can be done with the measured value?
- Are you really allowed to calculate with the measured values?

→ *Models, Metrics & Scales*

- What is “Quality”?

→ *Quality models*

- What does “Complexity” mean?

→ *The great obfuscation*

René Magritte: "The Treachery of Images", 1928–29.

Modeling – 1

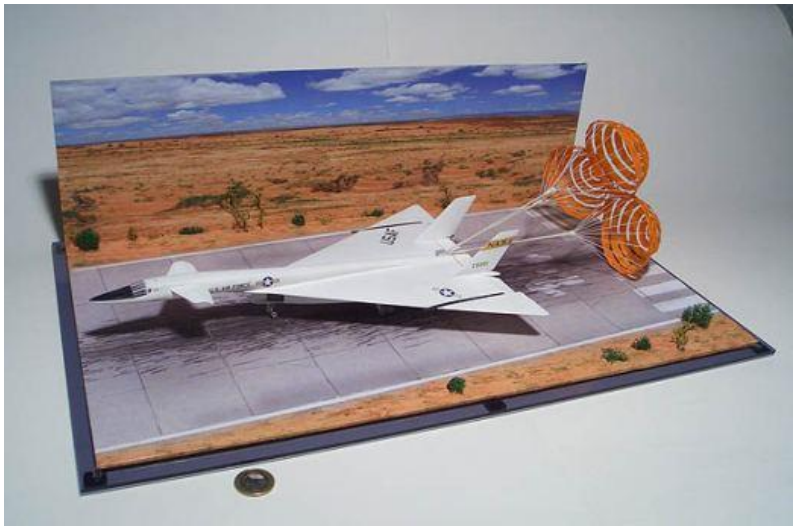
A model

- represents a system for a certain cause
- describes selected and relevant aspects of a systems
- neglects non-relevant aspects
- is easy to change
- gives different levels of abstraction

$\rightarrow$ a simplified image of the original

Original vs. Model

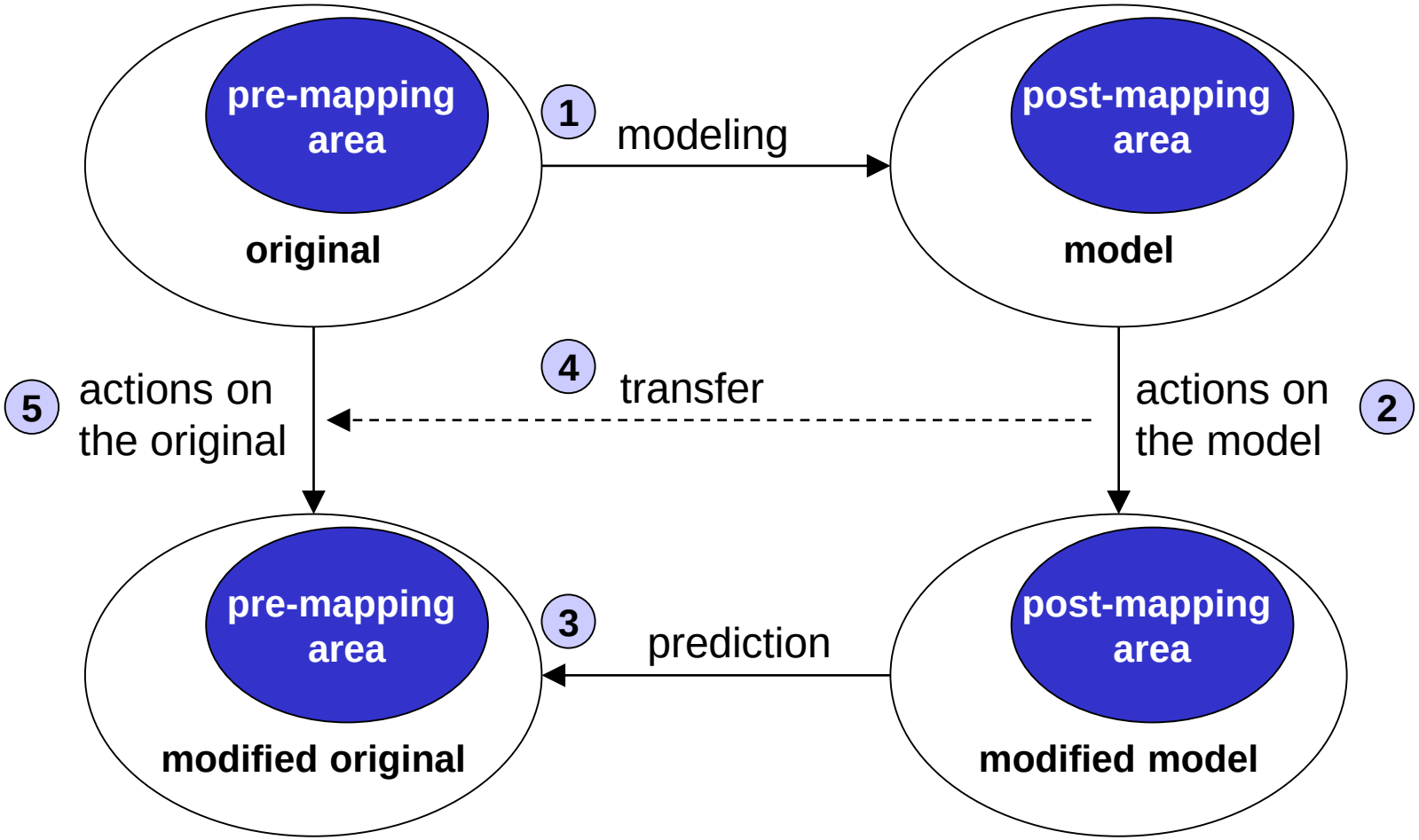
(a picture of the) Original



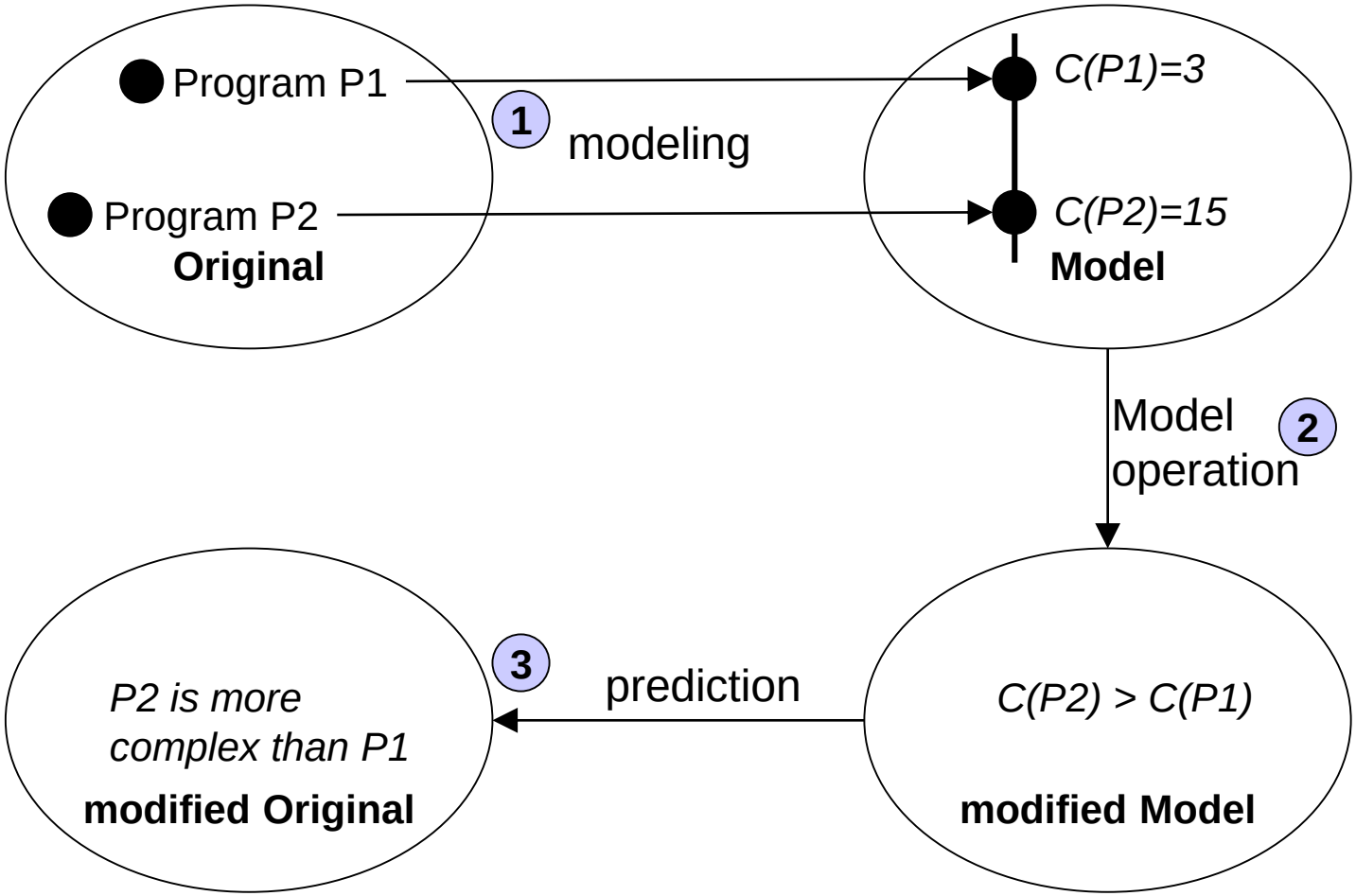
(a picture of the) Model

Source: http: //www.modellversium.de/galerie/8-flugzeuge-modern/2077-xb-70-valkyrie-lindberg.html

Modeling – 2


pre-mapping area
original
1
modeling
post-mapping area
model
5
actions on the original
4
transfer
actions on the model
2
pre-mapping area
modified original
3
prediction
post-mapping area
modified model

Modeling – Metrics are Models too



Scales

- *Nominal scale*: only classification, no operations
- *Ordinal scale*: Comparisons allowed
  C(P2) > C(P1) → „P2 is more complex than P1“
- *Interval scale*: Addition/Subtraction is allowed
  C(M1) + C(M2) → „the complexity of a system consisting of modules M1 and M2“
- *Rational scale*: Division/Multiplication is allowed
  C(S1) / C(S2) = 0,5 → „S2 is twice as complex as S1“

Modeling of Software Systems with Metrics (better name: *"Measured Values"*)

Reality	Metric
complex, difficult to access	handy, cheap
no simple ratings or comparisons	simple ratings or comparisons (depending on scale)
changes are difficult, expensive, risky	simple mathematical operations (and thus forecasts)

Classification of Metrics

- descriptive (i.e.: description of a current state) / prognostic (i.e.: forecast)
- product-related / process-related
- exact / estimated *(→ Estimations)*
- robust (i.e.: not to be faked) / forgeable (i.e.: can be faked)
- simple / derived
- makes statements about:
  - complexity / quality / errors / costs

Estimations: Guessing is a Metric too

Better to have an (not quite exact) estimation than some irrelevant numbers

Type	Result	Characteristics
Rough classification	Class (→ nominal scale)	■ **is** cheap ■ **not** differentiate, comparable, exact ■ **somehow** relevant, reproducible
Paraphrase	Text (→ no scale)	■ **is** differentiate, relevant ■ **not** exact, reproducible
Freehand estimation	Value	■ **is** differentiate, comparable, exact, relevant ■ **not** reproducible

Definitions

Quality Metric

(1) A quantitative measure of the degree to which an item possesses a given quality attribute.

(2) A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute.

IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990

Quality Models

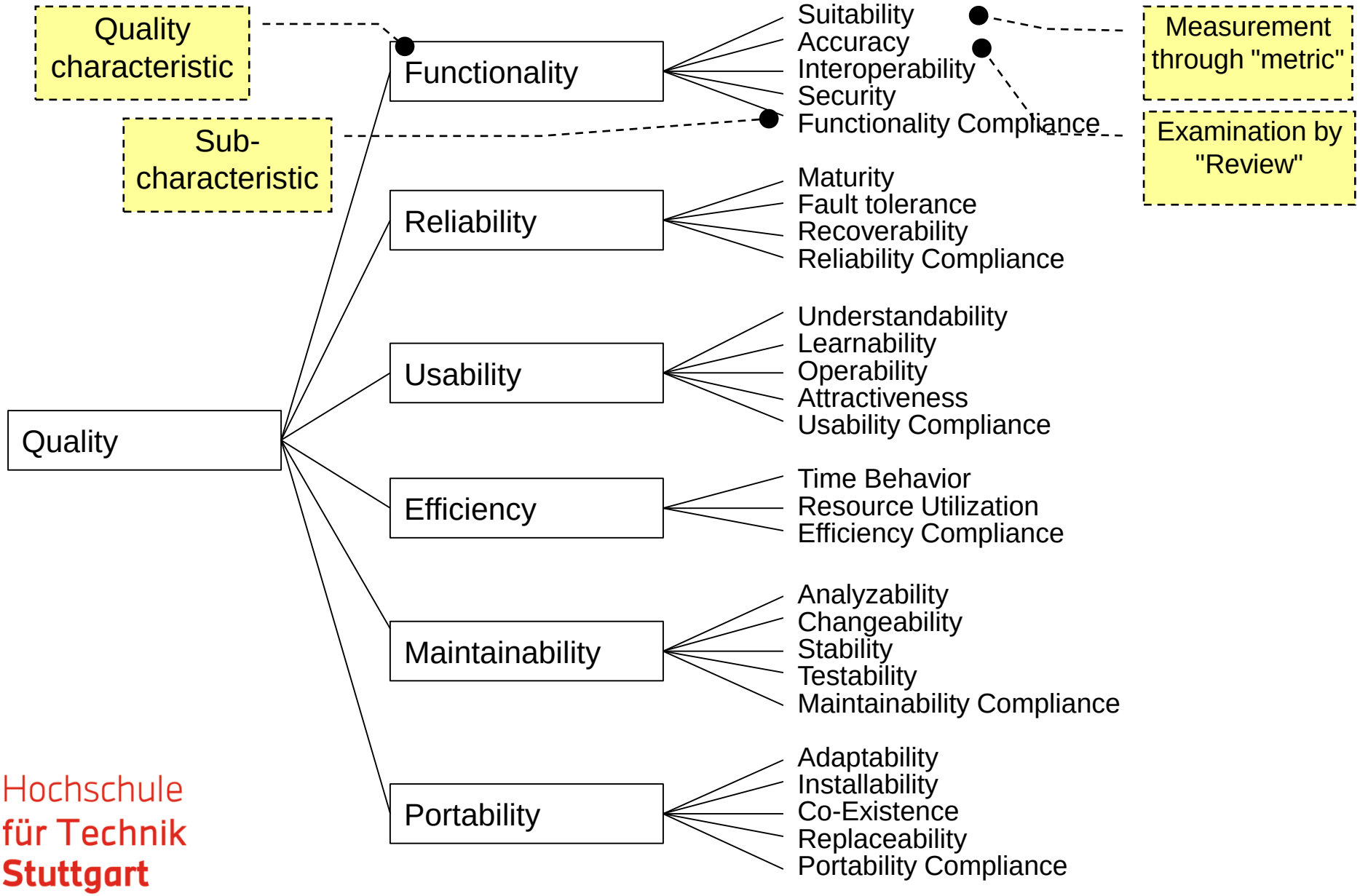
Software Quality: … As an attribute, software quality is

(1) the degree to which a system, component, or process meets specified requirements.

(2) The degree to which a system, component, or process meets customer or user needs or expectations.

IEEE 610.12 IEEE Standard Glossary of Software Engineering Terminology.

Quality tree according to DIN ISO 9126



Definitions

Complexity

(1) The degree to which a system or component has a design or implementation that is difficult to understand and verify.

(2) Pertaining to any of a set of structure-based metrics that measure the attribute in (1).

IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990

Complexity – The Great Obfuscation – 1

… According to several commentators, there is a distinction between the terms complex and complicated. Complicated implies being difficult to understand but with time and effort, ultimately knowable. Complex, on the other hand, describes the interactions between a number of entities. …

wikipedia.org: Programming complexity

Complexity – The Great Obfuscation – 2

- There is no clear definition.
- Everybody has a feeling of it.

→ Everybody can declare a complexity metric on anything.

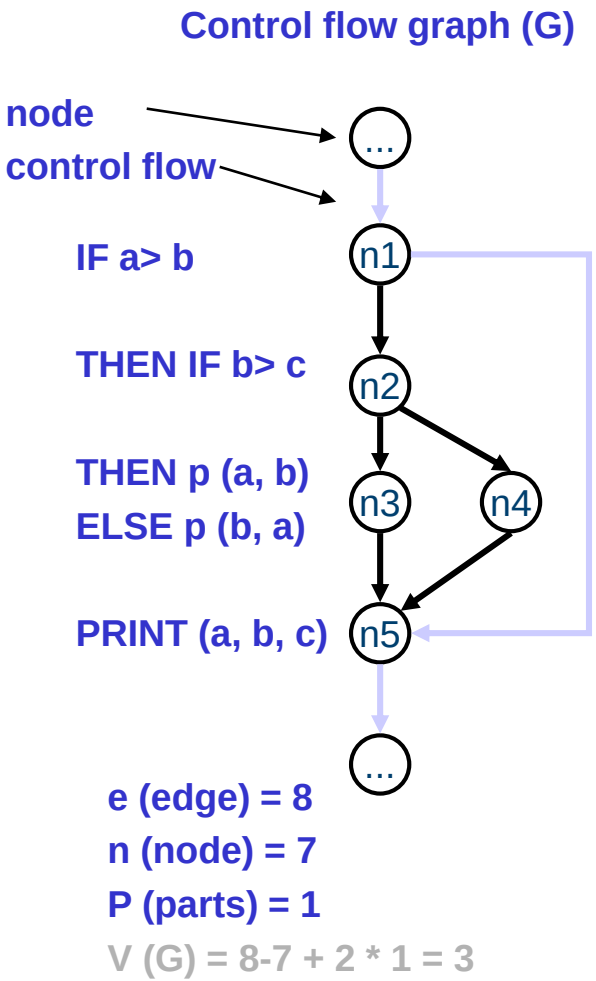
→ Nobody can prove that something is not measuring complexity.

- E.g. “Cyclomatic Complexity” measures actually the number of independent paths in a code and declares the number to be the complexity of it.

How to Come up with a Sensible Metric

1. State the goal you want to achieve
2. Decide what object to measure: process or software?
3. Decide what state to measure: past or future?
4. Make a Quality Model of your object.
5. Decide which attributes to measure.
   - also: how to measure: estimation or precise?
   - also: what scale?
6. Define how to calculate: direct or derived?
7. Do measurement over a certain time
   - validate with measurements of the same attribute
   - calibrate measurements
   - observe trends

E.g. Cyclomatic Complexity (McCabe, 1976)



1. Goal: Measurement of maintenance difficulty
2. Object: Source Code
3. State: post-mortem
4. Model: code as graph
5. Attributes: graph nodes and edges (precise on a rational scale)
6. Derive: v(G)= edges – nodes +2 * parts (parts usually 1)
7. Validate: e.g. let programmers estimate the complexity and compare to v(g)

Measuring Design

Problem

- Automated metrics are working on code
- There is no code at the start of the design

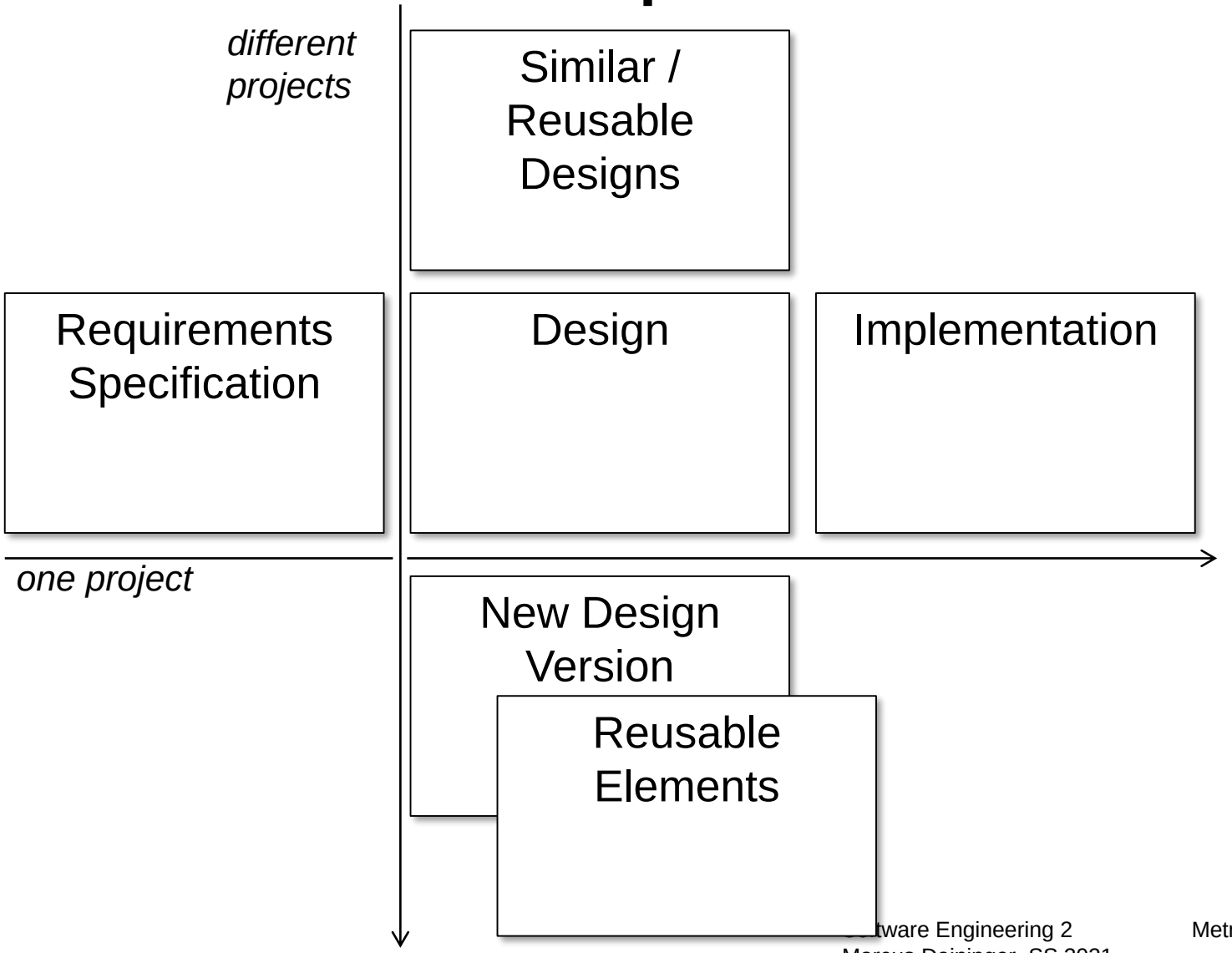
→ Design-Metrics based on Code only give a post-mortem-view

- A HP-Designer: „Design-code metrics are only used for maintenance“

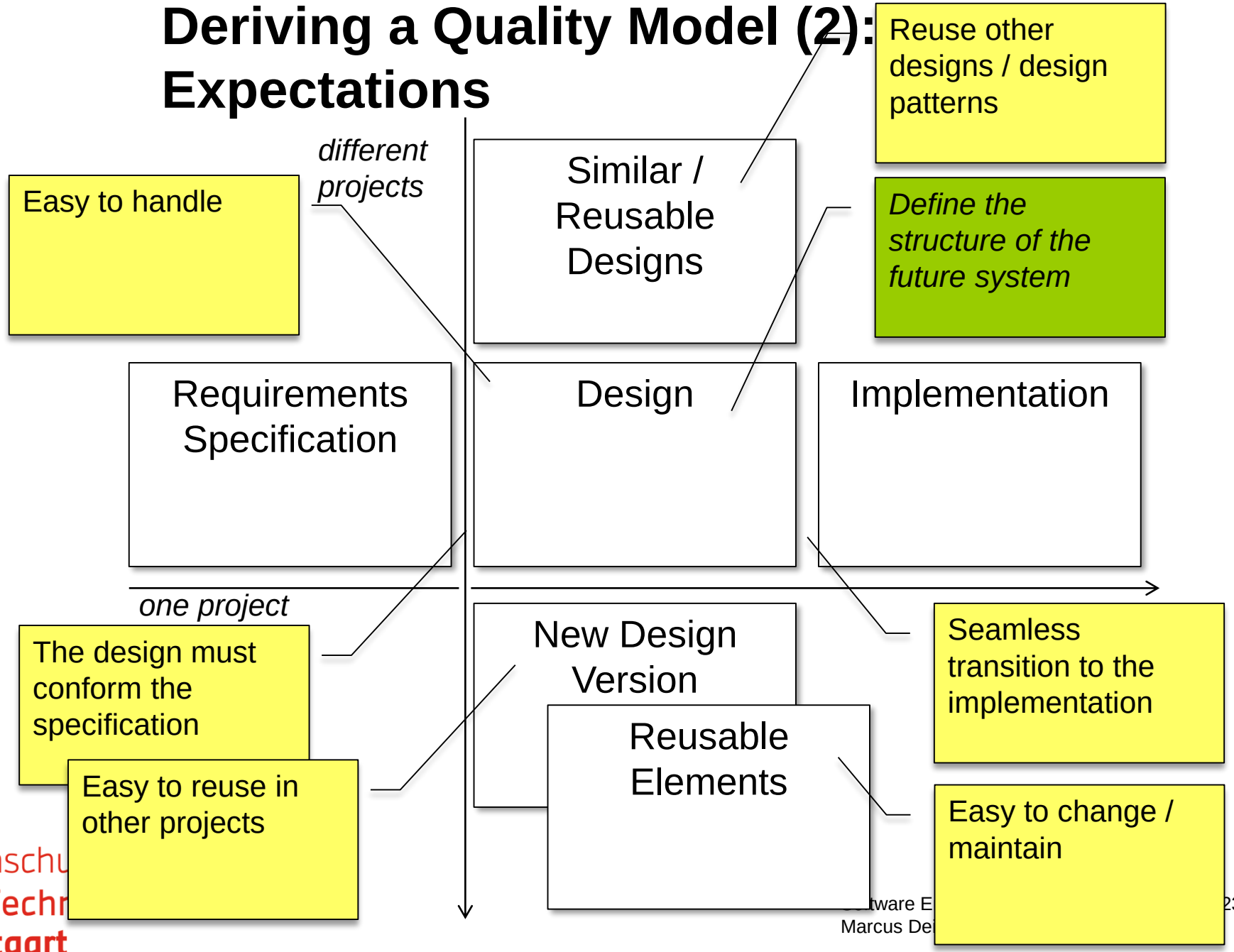
What to do instead?

- Build a quality model for design
- Select attributes of interest
- Measure the artifacts so far created – which seem correlated to the attributes

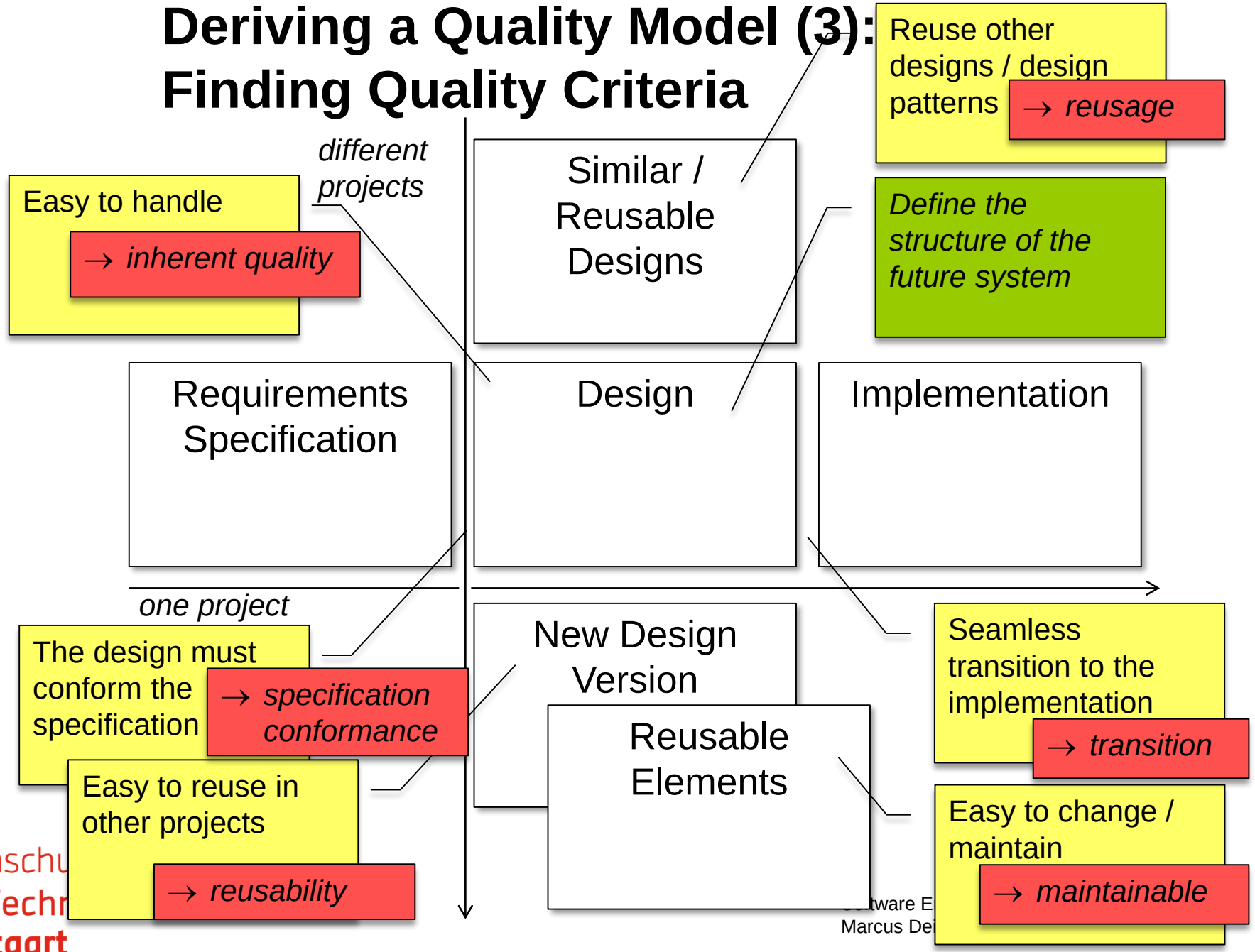
Deriving a Quality Model (1): Design in the Software Development



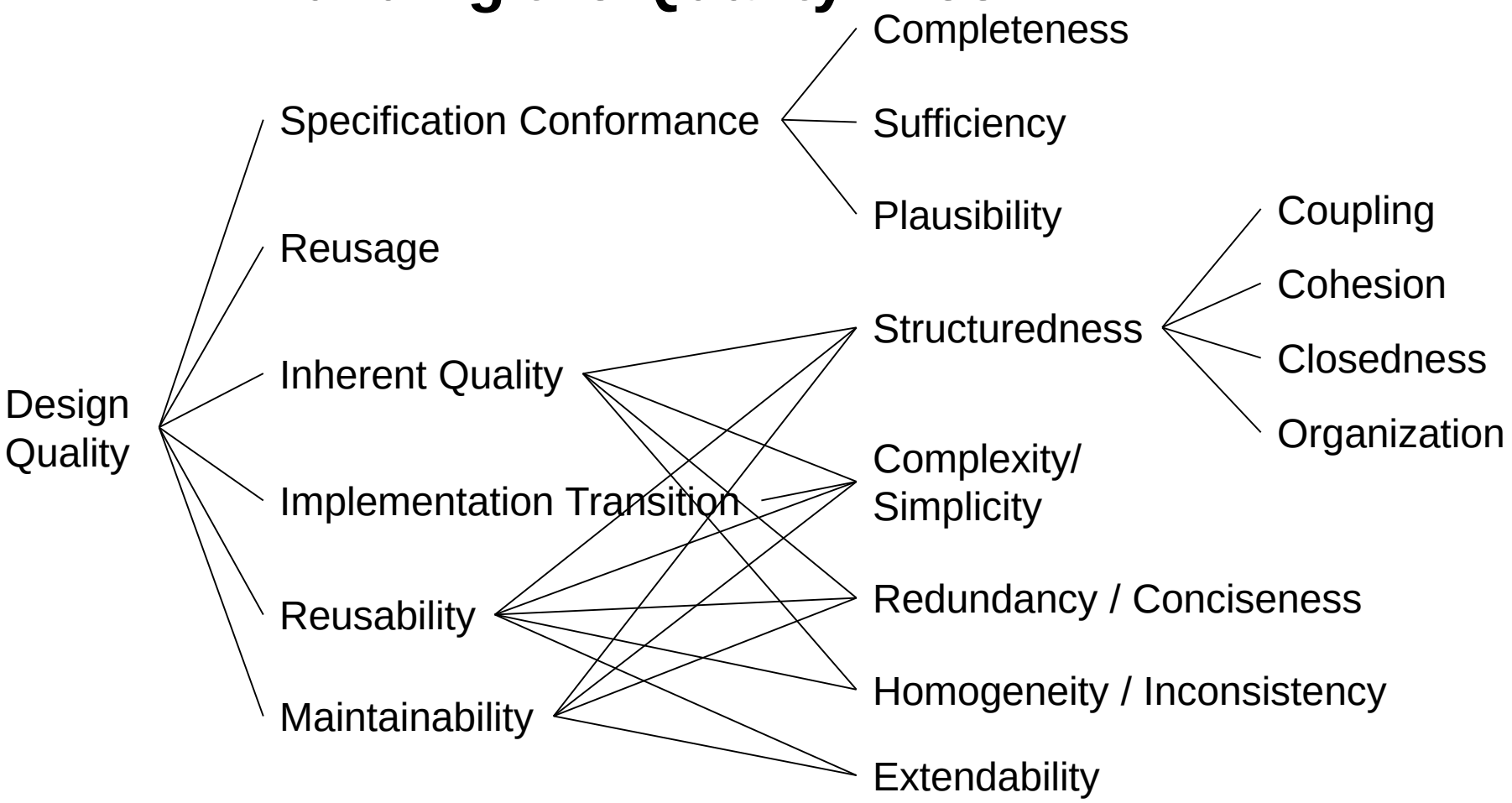
Deriving a Quality Model (2): Expectations



Deriving a Quality Model (3): Finding Quality Criteria



Deriving a Quality Model (4): Building the Quality-Tree



How to Set-up/Use Metrics

- define precise and project specific, calibrate over time/projects
- define a simple set of basic metrics
  - Count/Record the previous elements + effort/errors
- derive a few effort metrics (…/effort) and quality metrics (…/error)
- set them in relation to
  - other projects
  - the overall project average
- observe over time
  - detect anomalies and forecast trends
  - investigate anomalies with special tools or reviews
- take results seriously, bear consequences

"There's no such thing as a free lunch!"

Elements, which may be counted

Code

- Number of packages
- Number of classes
- Relationship between the classes

→ *Martin-Metrics*

Project

- Effort
- Duration
- Errors found (in Reviews)

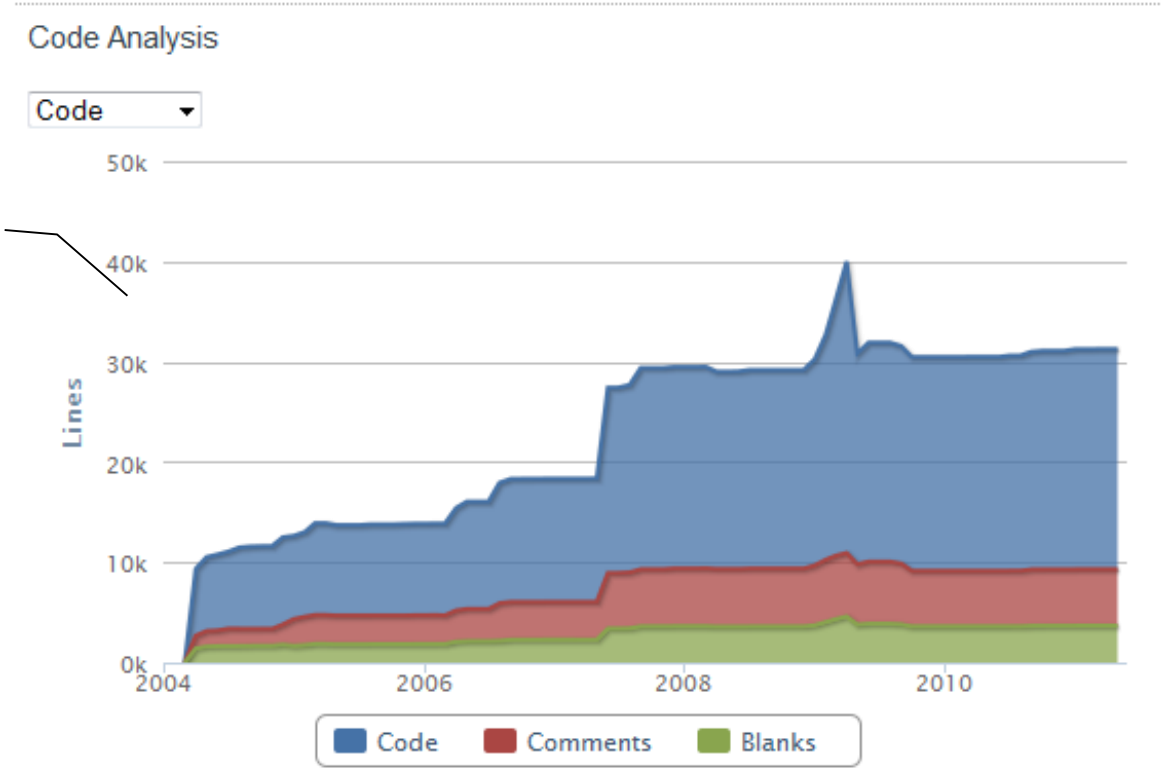
Spec

- Number of business functions
- Number of actors
- Volume of user interaction
- Volume of DB-interaction

→ *The last two resemble the "Function Point" measurements*

Best Practice

- Better collect a simple set of metrics over time
  - e.g.: loc, errors, cost



Development of code over time.

Source: https://www.ohloh.net/p/JGR