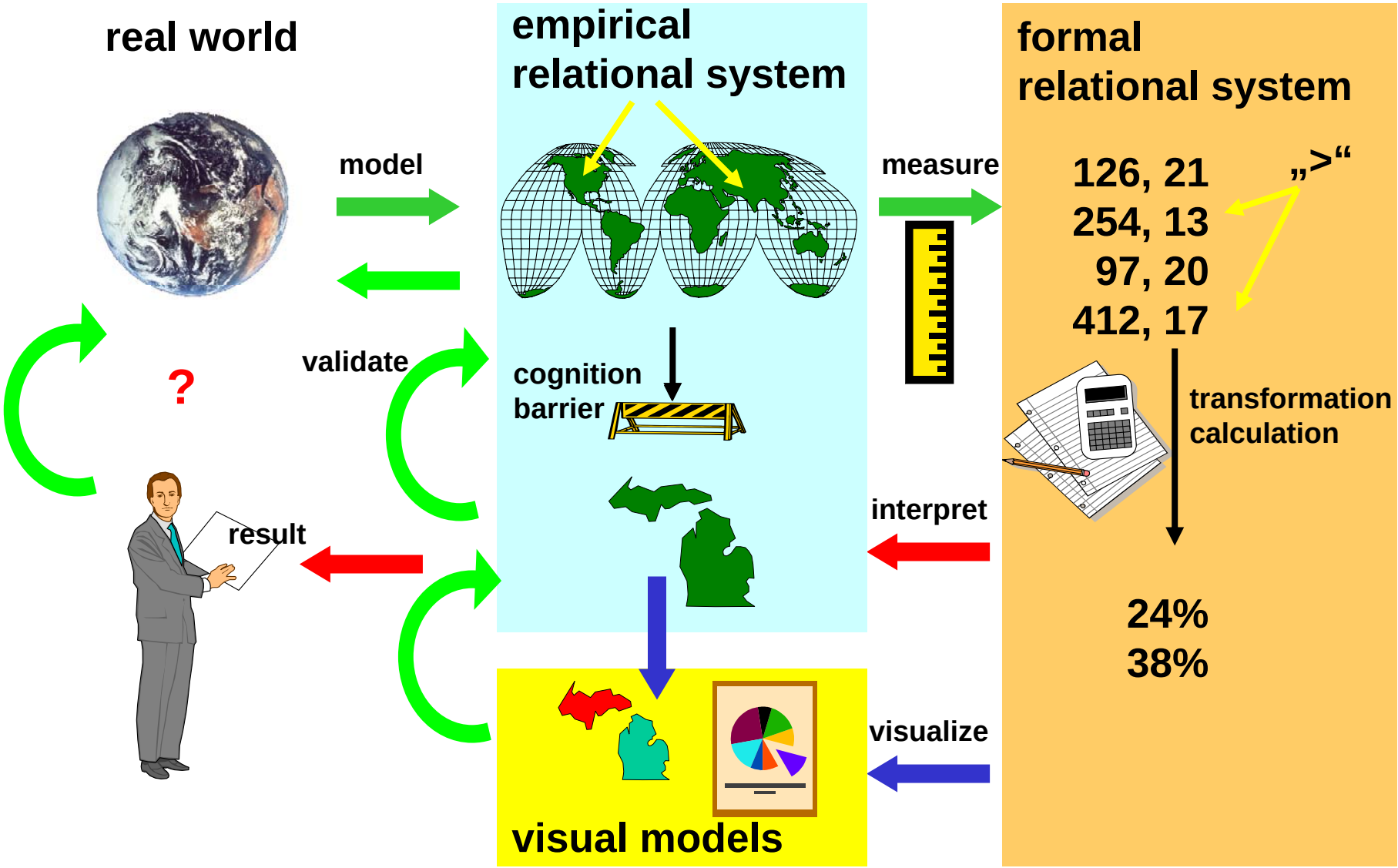
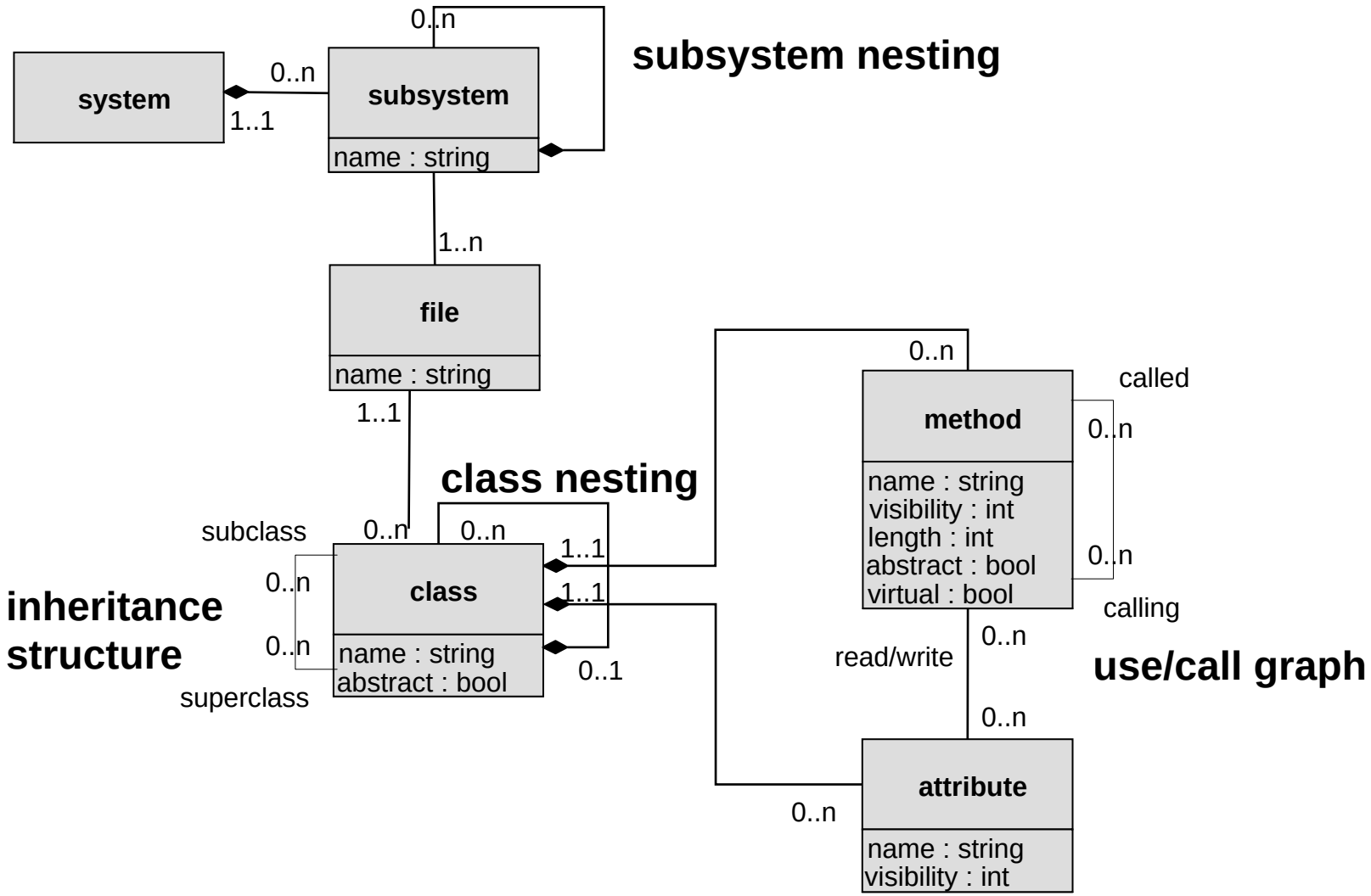
Measurement-based Product Analysis



Example: Basic Structural OO Product Abstraction



Software Produkt-Metriken

Kategorien:

- Größe — Anzahl der Komponenten (K) einer Einheit
- Kopplung — Anzahl der K, die eine K kennt, verwendet oder voraussetzt
  (Aufruf, Zugriff auf Attribut, etc.)
  ausgehend (afferent), eingehend (efferent)
- Kohäsion — Intensität des Zusammenhangs der K innerhalb einer Einheit
- Komplexität — ??? algorithmisch / kognitiv ???

Einige bekannte Software-Metriken (1)

- Halstead
  - PL/1, Fortran: complexity for writing/maintaining programs
  - Programm = Menge von Operationen + Menge von Operanden
  - Basis-Metriken:

n	# verschiedener Operatoren
m	# verschiedener Operanden
N	# Operatoren
M	# Operanden

  - abgleitete Metriken:

Programm-Länge	$L = N+M$
Volumen	$V = L*\log_2(n+m)$
Komplexität	$K = (n/2)*(M/m)$
Aufwand	$E = V*K$
etc.	

nie validiert! genügt nicht der Messtheorie!

Einige bekannte Software-Metriken (2)

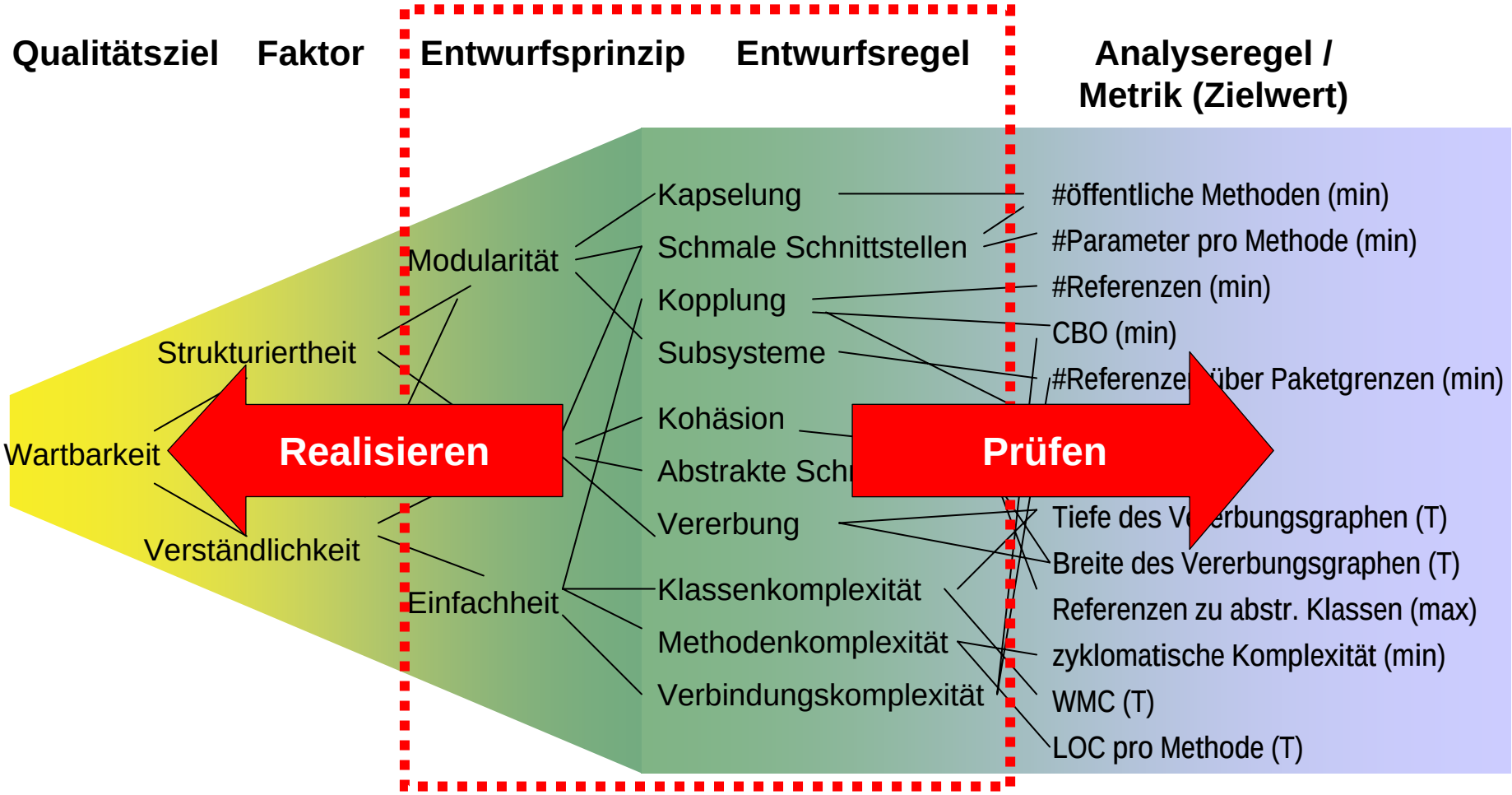
- McCabe „Cyclomatic Complexity“ (´76)
  - Ziel: Komplexität für Wartung und Verstehen messen
  - Idee: Struktur berücksichtigen; Kontrollflussgraph (CFG)
  - Programm Graph P = (E,V)
    - v $\in$ V meist Funktionen
    - e $\in$ E Kontrollflussabhängigkeiten
  - CC(P) = |E| - |V| + p (p: Zshg.-Komponenten)

  - Gegenbeispiel s. 24.06.04
  - empirische Korrelation
  - für OO nicht geeignet

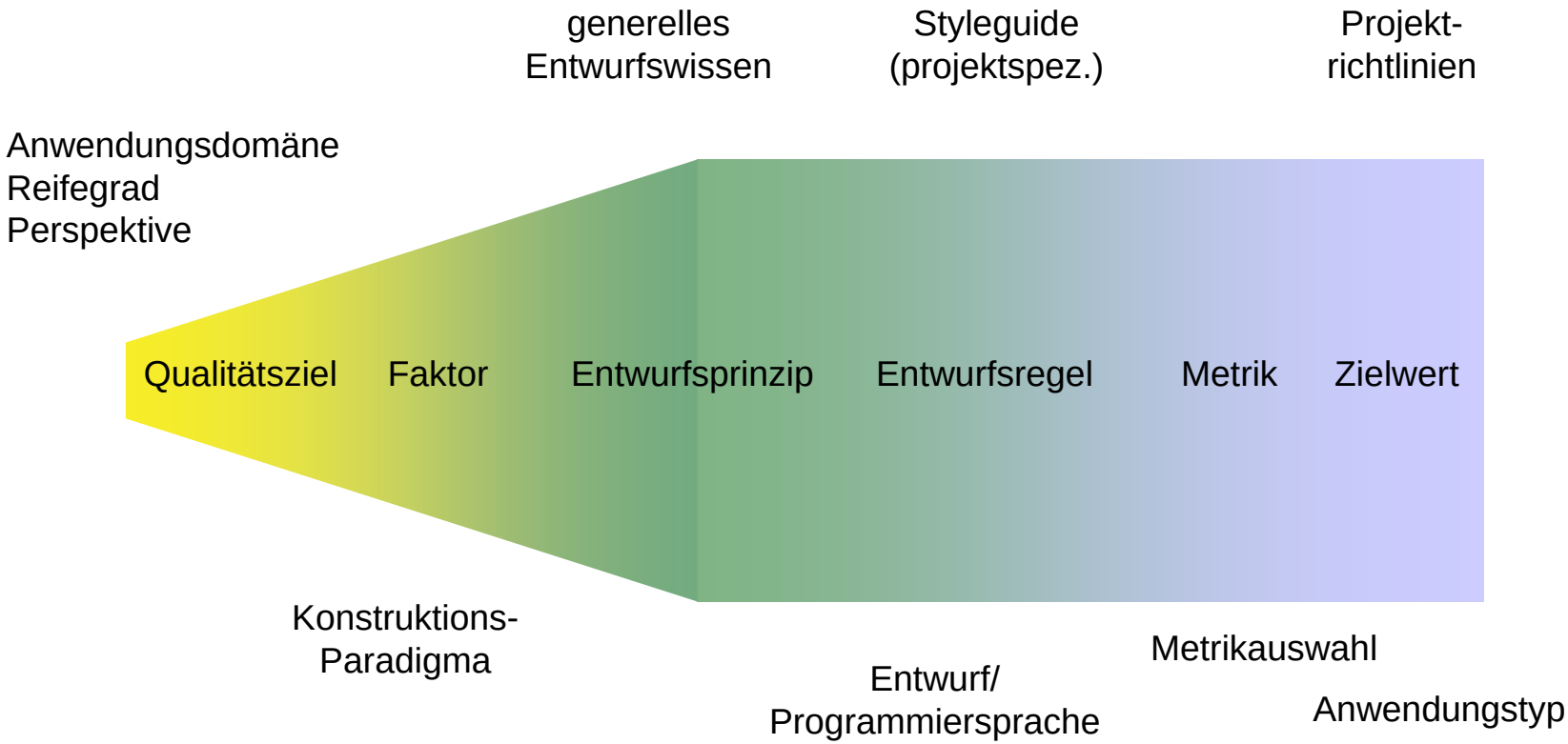
OO Metriken

- DIT: depth of inheritance tree
- NOC: number of subclasses
- CBO(c): coupling between objects
  # der benutzten Attr./Methoden anderer Klassen
- RFC(c): response for class
  # Methoden von c + # aller benutzen Methoden anderer Klassen
- LCOM(c): lack of cohesion
  ~ wie misst man, ob Klasse c einen abstrakten Datentyp implementiert „eine Klasse – ein Konzept“?

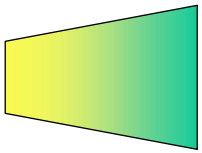
Beispiel eines 'OO Wartbarkeit' QM



Einflüsse auf das Qualitätsmodell

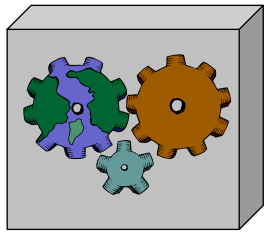


Example from a Case Study



- **DFS quality goals:**
  - **Maintainability**
  - **Level of object-orientation**
  - **Portability**
  - **Dependability**

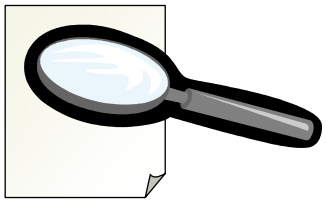
focus of analysis



- **OO models a system as a set of collaborating objects**
- **Objects have a state and an interface**

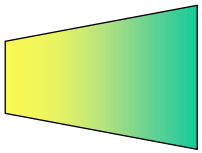
- **OO Design Principles**
  - **Bundling**
  - **Encapsulation**
  - **Inheritance**
  - **Polymorphism**

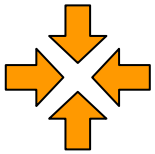
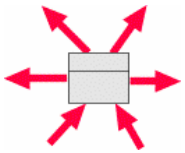
Apply principles in favour of quality goals



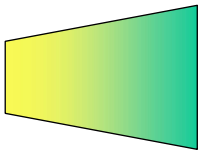
Check, whether principles have been applied properly

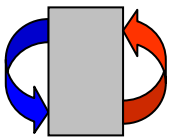
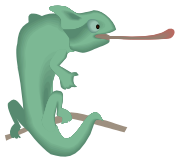
Quality Criteria and Metrics



Design Concepts	Metrics (target)
 **Bundling:** everything is a class!	▪ Usage of class/struct keywords (use class!) ▪ number of global variables/functions (min)
 **Encapsulation:** ▪ hide data representation ▪ make encapsulation explicit ▪ encapsulate platform dependent parts ▪ Low coupling, high cohesion	▪ Usage of keywords private/public/... ▪ Number of public attributes (min) ▪ Usage of abstraction layers
 **Coupling: „degree of dependencies“** ▪ Directions: out-/ingoing ▪ Kinds: call/attribute/inheritance	▪ attribute couplings (min) ▪ bidirectional couplings (min) ▪ access to attributes of superclass (min) ▪ CBO (min)

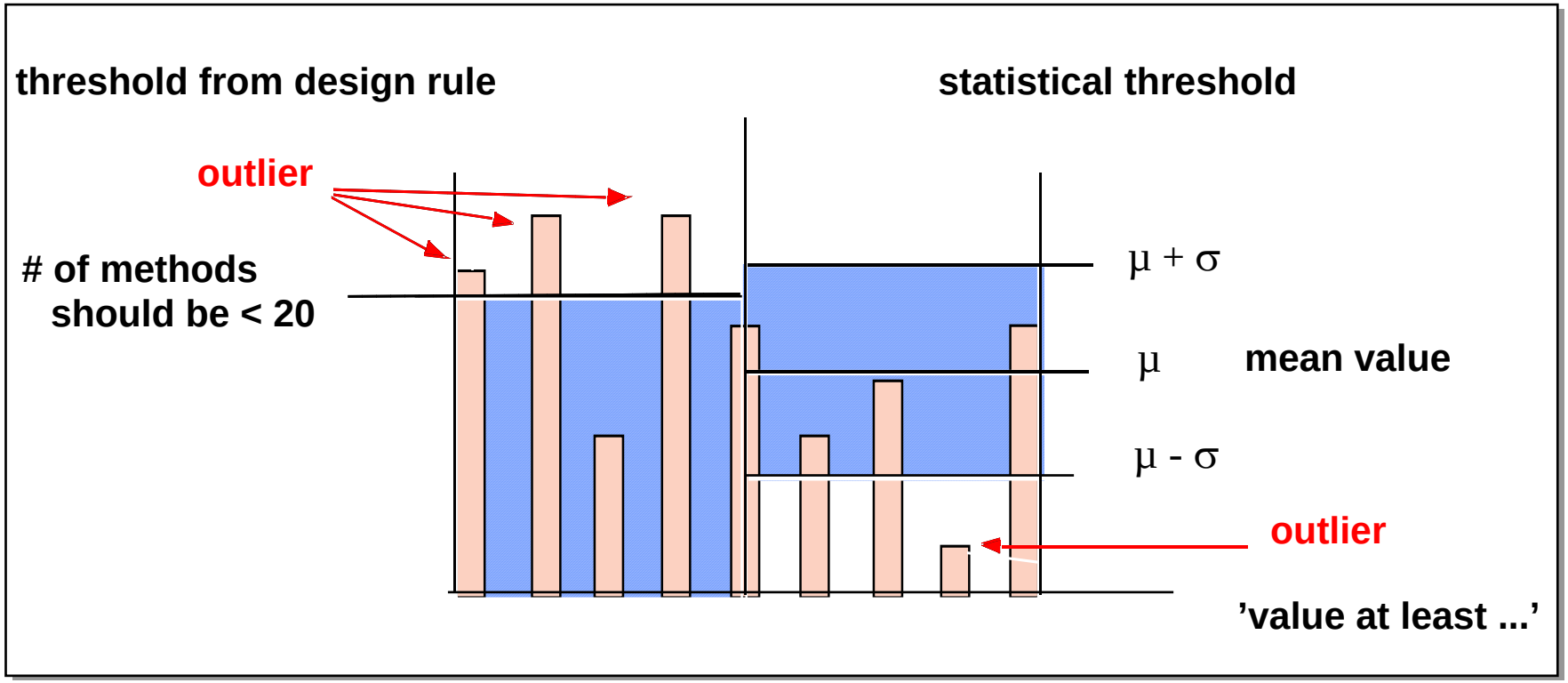
Quality Criteria and Metrics



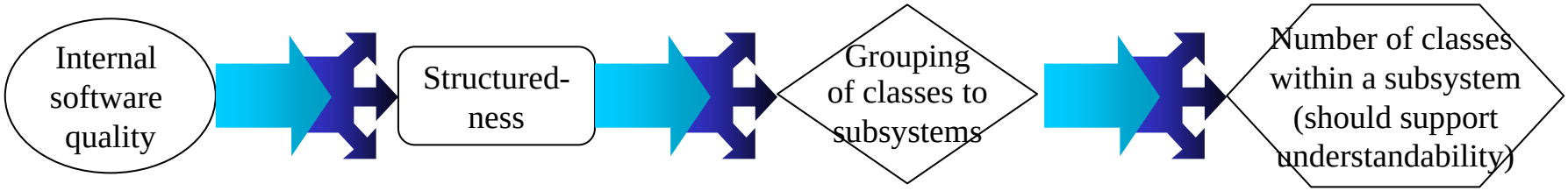
Design Concepts	Metrics (target)
 **Cohesion: „degree of togetherness“** ▪ Levels: package/class/method ▪ One class = one responsibility	▪ No. of attributes/methods per class (T) ▪ LCOM (min) ▪ Number of private method calls (max)
 **Inheritance:** ▪ Express specialisation ▪ Factor out common class elements	▪ No. of abstract classes (max) ▪ DIT (T) ▪ NOC (T)
 **Polymorphism: „many forms“** ▪ Abstract ▪ One class = one responsibility	▪ Number of overwritten methods (max) ▪ Number of enums and switches (min)
 **Coding style:** ▪ Central places for constants ▪ Simple control flow	▪ Distribution of string constants (min) ▪ CC (min) ▪ Number of if/switch per unit (min)

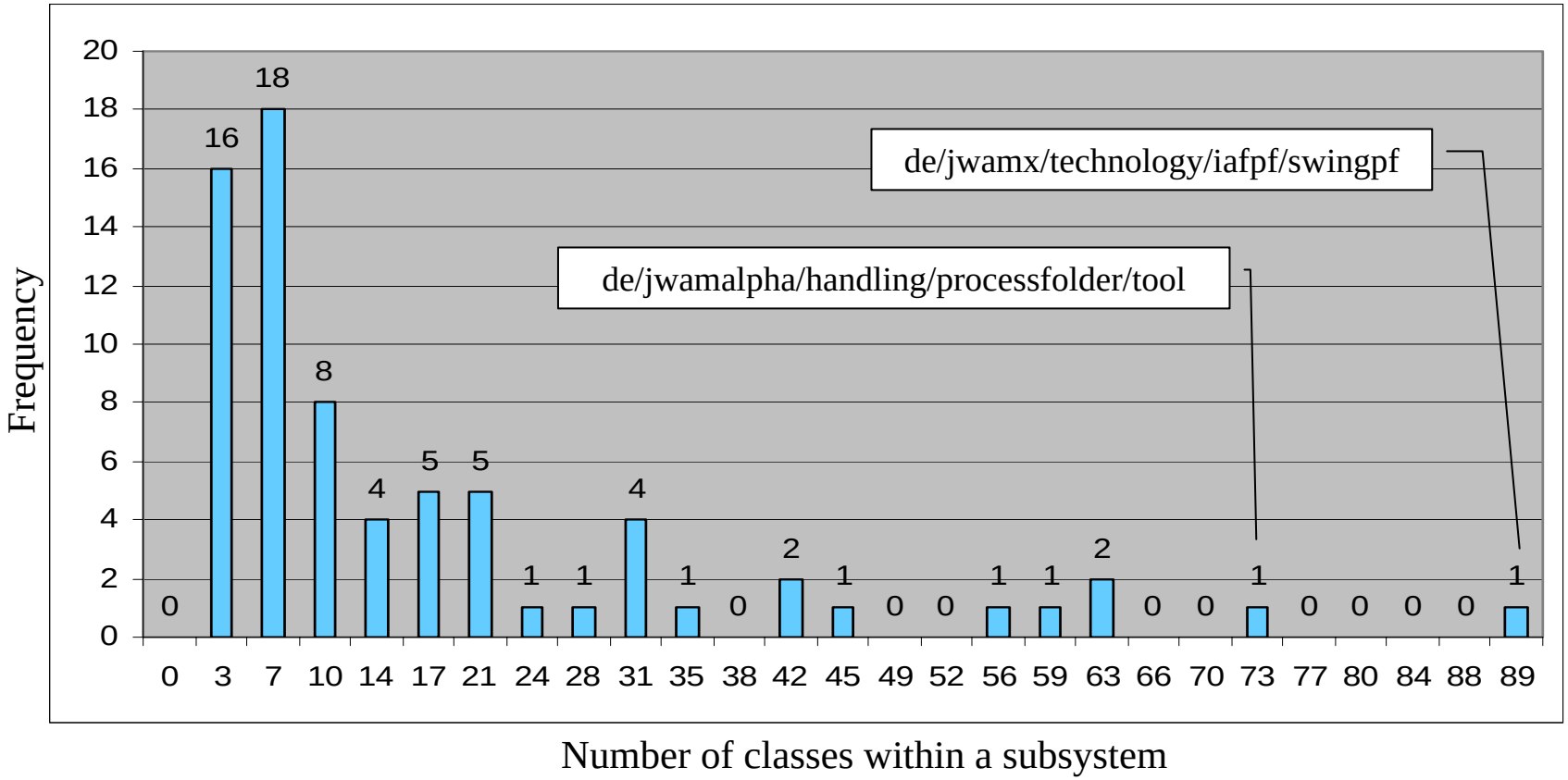
Interpretation of Metrics Values

- value filter functions
- definition of threshold values (design rules / statistics)

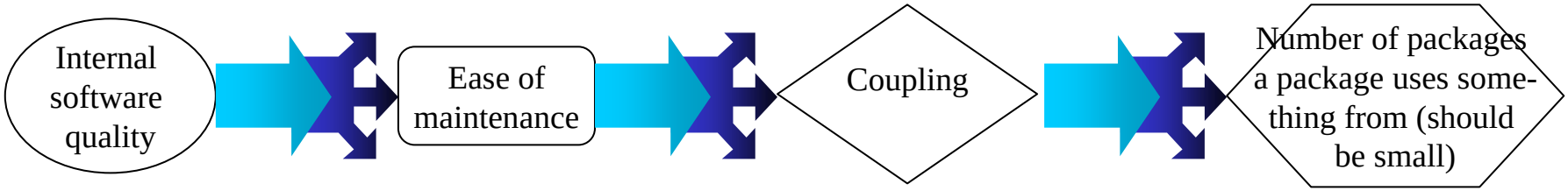


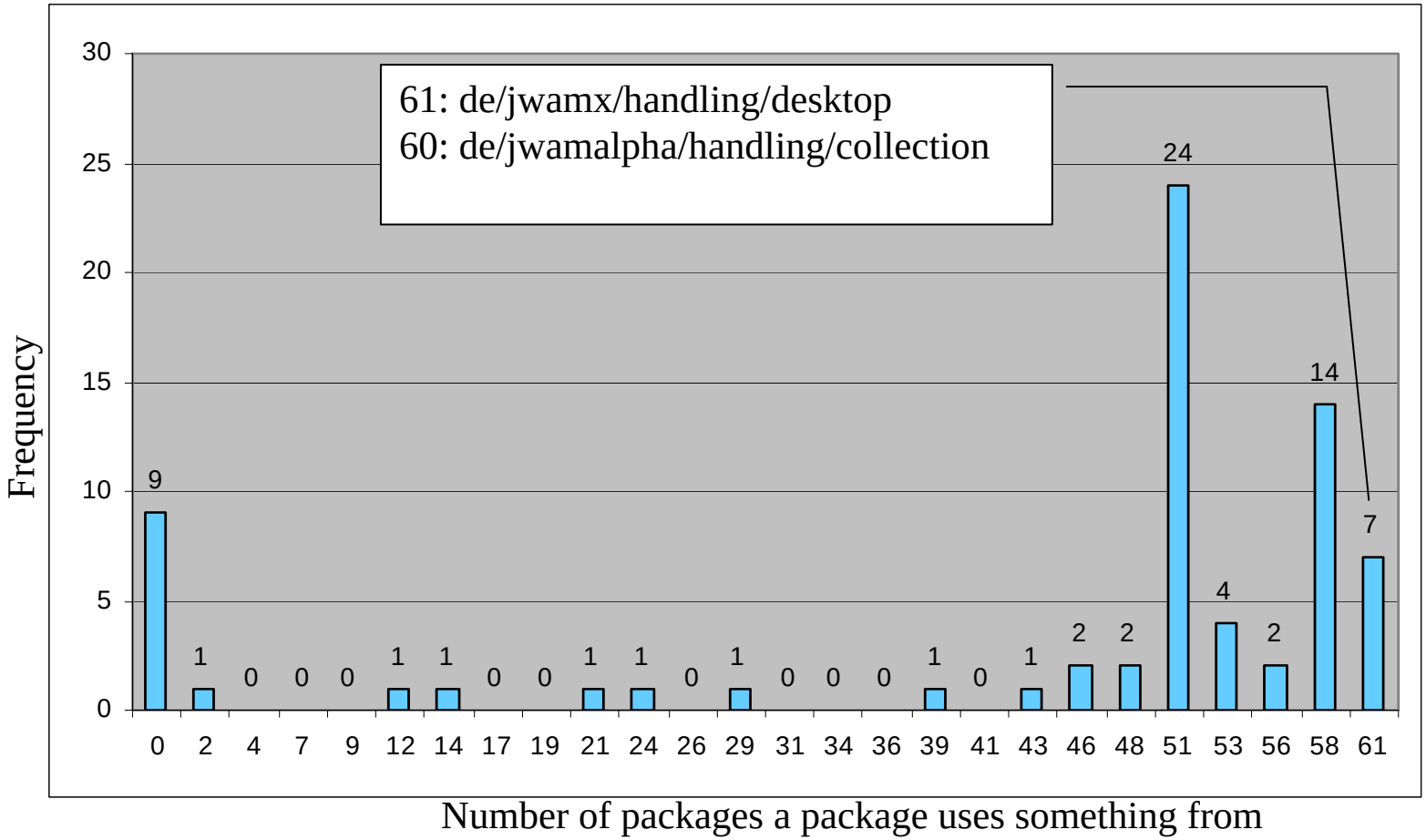
Quality assessment (package structure)



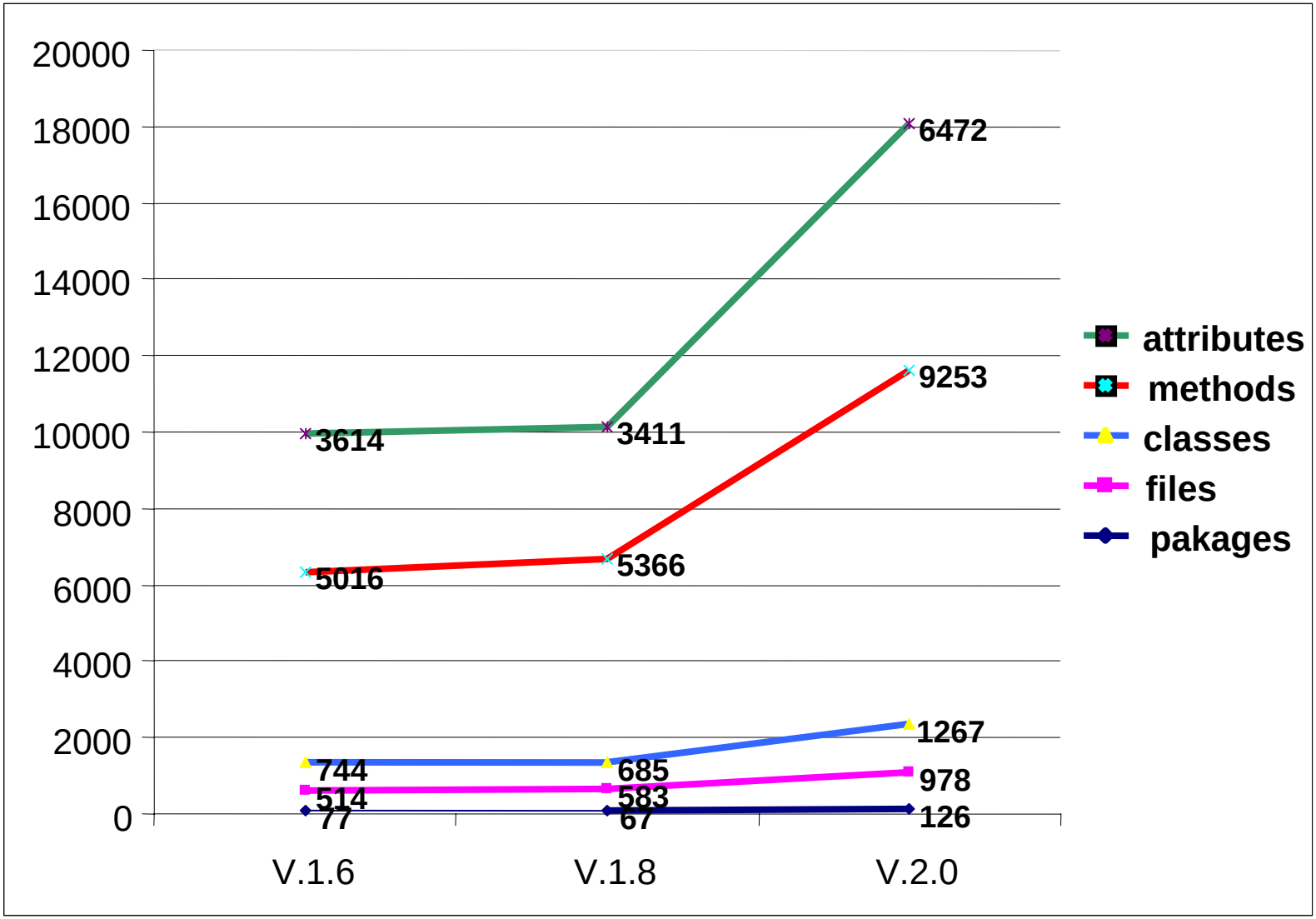


Quality assessment (coupling between packages)

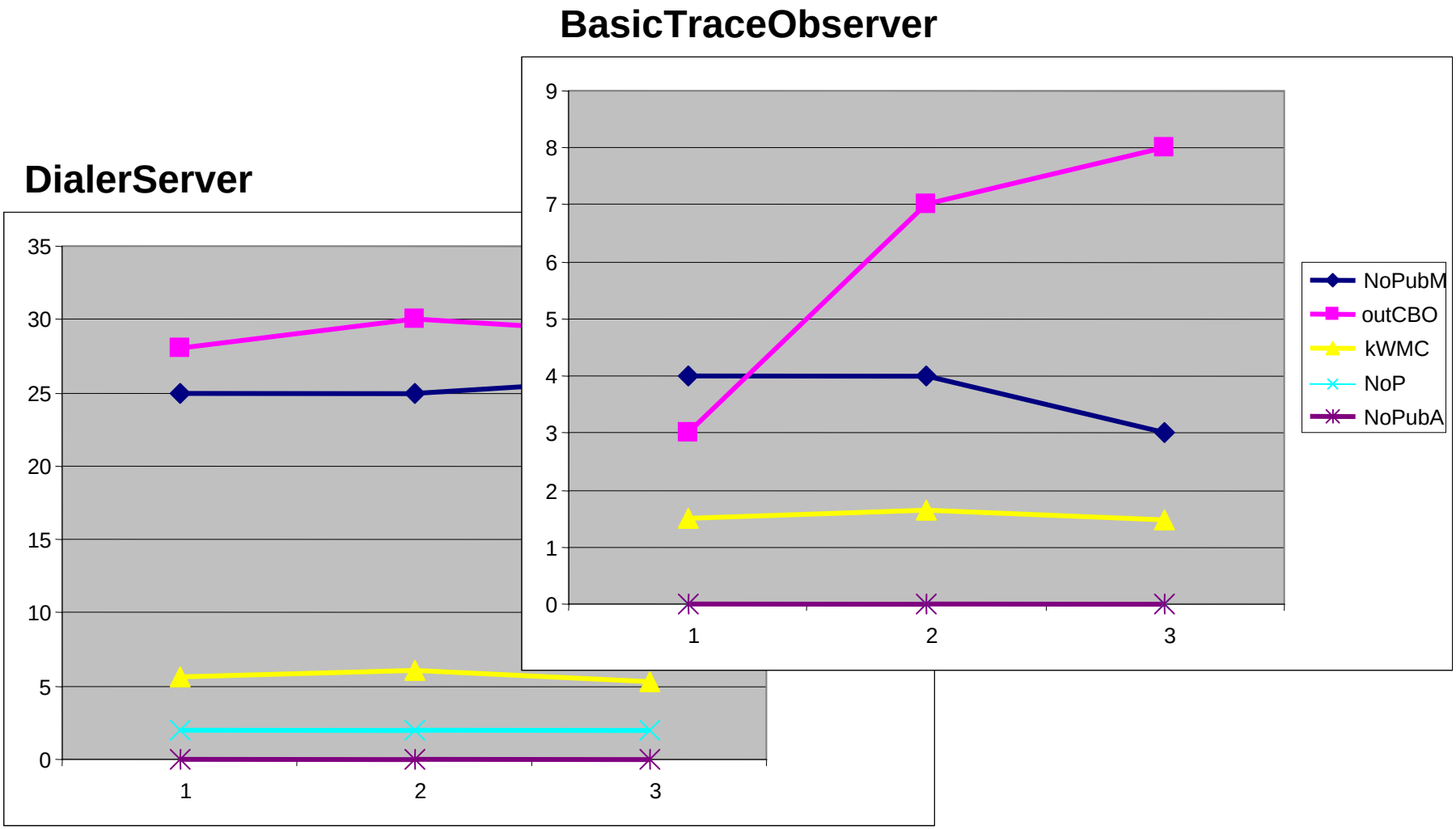




Trend Data for 3 System Releases

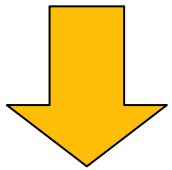


Trend Data for 2 Classes

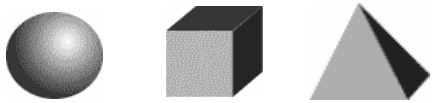


Visualisierung komplexer Strukturen

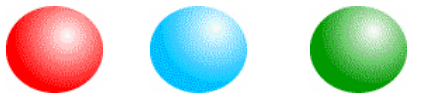
- Objekttypen (Paket, Klasse)
- Eigenschaften (Messwerte)



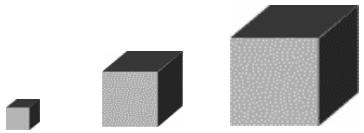
Formen



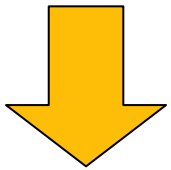
Farben

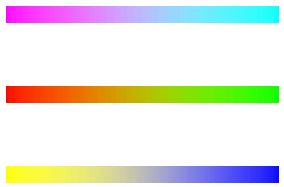


Größe



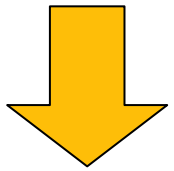
- Beziehungen zwischen Objekten (Vererbung, Aufruf, Benutzt) und deren Richtungen

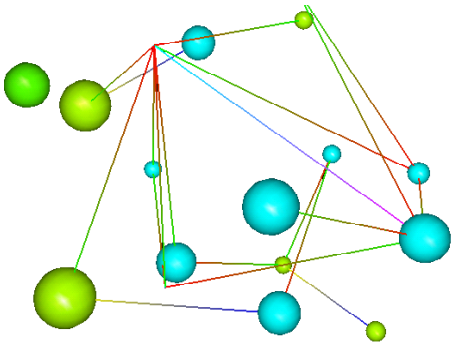




Kanten mit verschiedenen Farbübergängen

- Ähnlichkeit von Objekten (bzgl. Beziehungen mit anderen Objekten)

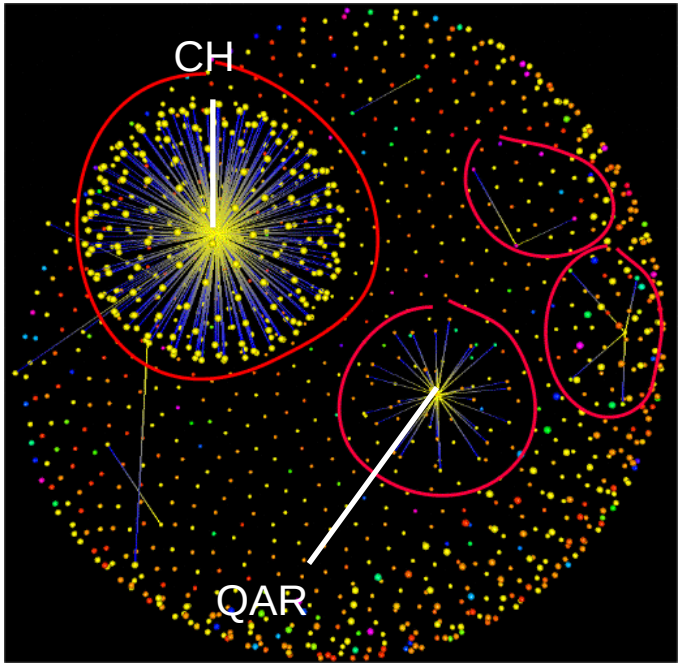




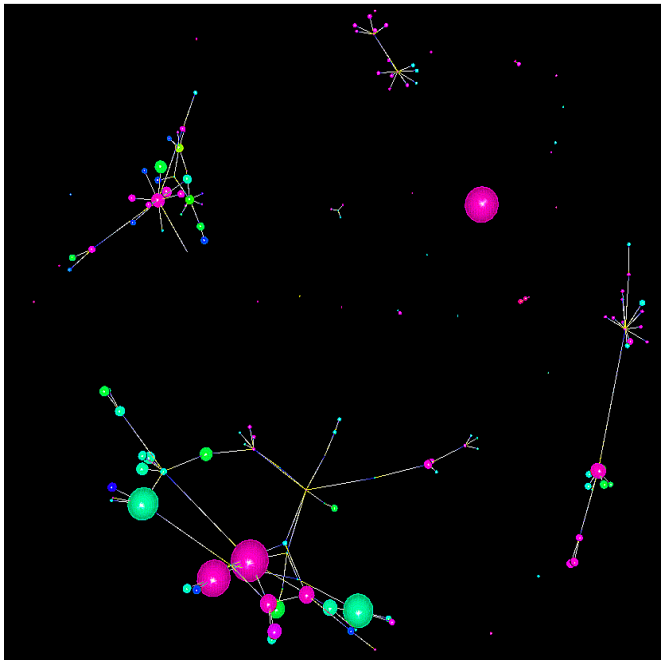
Layout (ähnliche Objekte räumlich nah)

3D Graph Vererbung

Untersuchtes System



Anderes System

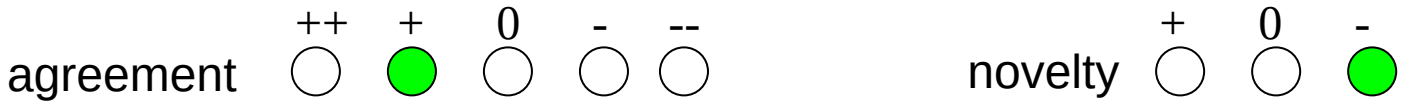


- Nur 8 Klassen vererben, 2 Klassen haben 250 bzw. 30 abgeleiteten Klassen
- Vererbung gering genutzt – geringe Tiefe, extreme Breite
- Risiken:
  - Chance, Ähnlichkeitsbeziehungen aus Problembereich im Code auszudrücken wenig genutzt
  - Dadurch erschwertes Verständnis

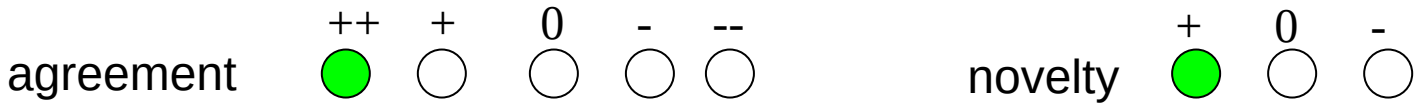
Metapher: Stadt/Landschaft

Diagnoses & Recommendations
(two examples from analysis of JWAM 1.4)

Lines of Code: The file *HtmlStandardWriter* seems too large (1277 LOC).



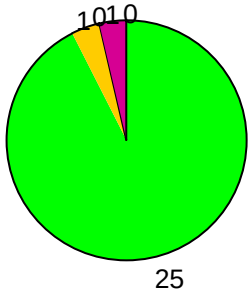
Number of public Attributes: The class *ConfigurationStandard* contains 18 public attributes. Because they are used only by methods of this class the visibility should be reduced to protected at least. The same should be applied to class *ClassUseMapper*.



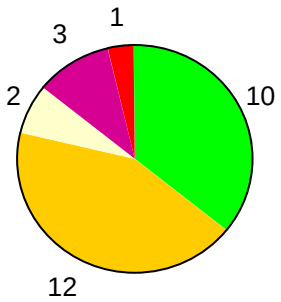
Evaluation of our results:

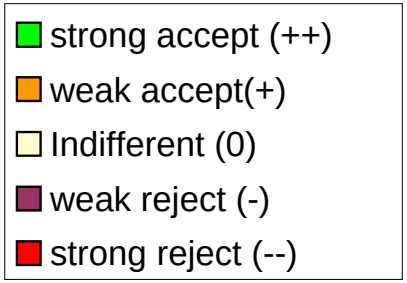
JWAM 1.4:

27 diagnoses



28 recommendations



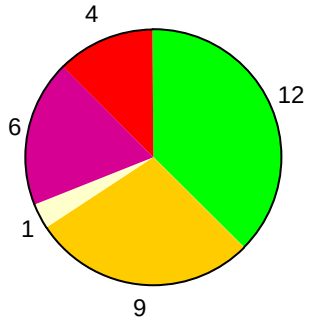


➔ the most relevant critical situations got repaired in JWAM 1.5

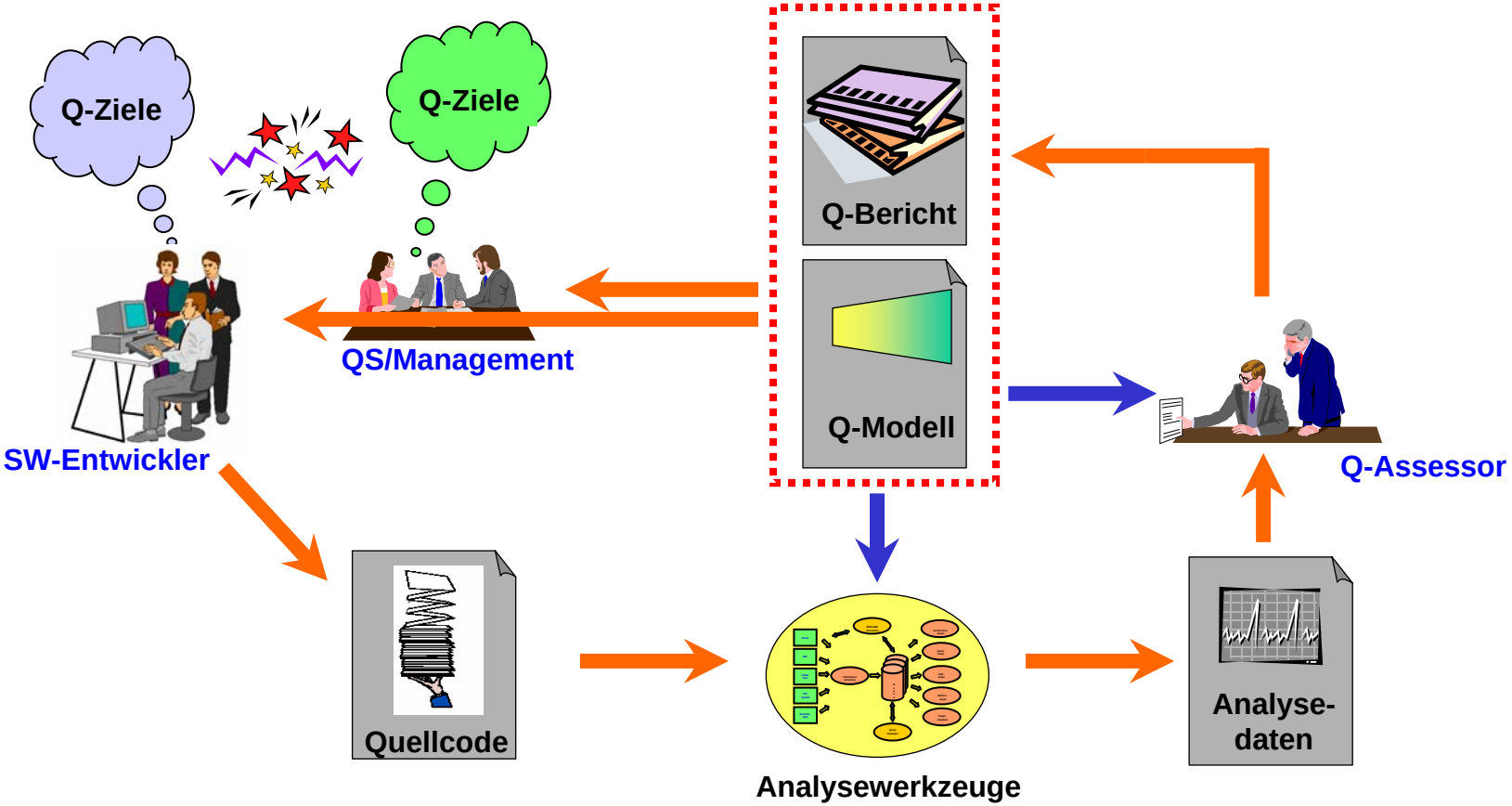
JWAM 1.5:

36 diagnoses

32 recommendations



Szenario 1: Initiales Q-Assessment



Szenario 2: Review-Unterstützung