Software Engineering 2 – Metrics. Part 1

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

Metrics …

» The Answer to the Great Question …
of Life, the Universe and Everything ...
is... « said Deep Thought, and paused …
» … is ... Forty-two. «

Douglas Adams,
The Hitch Hiker's Guide to the Galaxy

Typical Questions during Software Development

- How good is the software in terms of structure, understandability, efficiency, accuracy, ease of use, ease of modification?
- Is there a overall logical architecture? Can parts of the system be replaced by newer technology? Can the software be extended without deteriorating the quality?
- Can the software be tested? Can the software be maintained at reasonable costs? Are there any individual components reusable?
- Are design rules respected in the implementation? Are technology standards used throughout?

Definitions

A Function
Metric: Element → Number

Metric

A quantitative measure of the degree to which a system, component, or process possesses a given attribute.

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

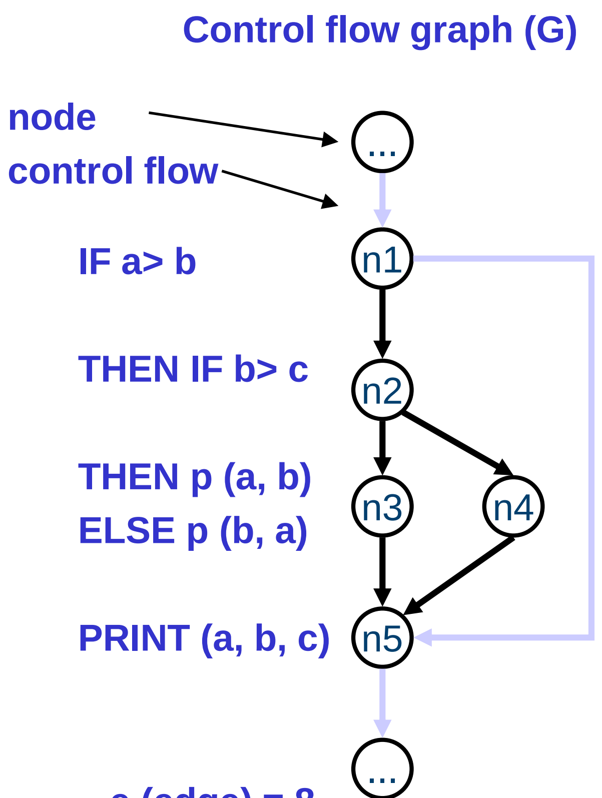
Classification

- Counted Elements
  - Lexem-based: counting of lexems (e.g. eol)
  - Structure-based: counting of syntactical elements (e.g. decision points)
  - Event-based: counting of events (e.g. errors)
  - Rating-based: “measurement” by (human) rating
- Focussed Units
  - Unit-based: Counting the elements of a single unit
  - Relationship-bases: Counting relationships between elements (typically structure-based)
  - Project-based: Counting project metrics

Simple Metrics

- Lines of Code: Unit-/Lexem-based
- Defects: Unit-/Event-based
- People/Time/Cost: Project-based

Cyclomatic Complexity (McCabe, 1976)



e (edge) = 8

n (node) = 7

P (parts) = 1

V (G) = 8-7 + 2 * 1 = 3

- Unit/structure-based measurement
- Counts the nodes and edges of the control flow graph
- v(G) = edges – nodes +2 * parts (parts usually 1) → Number of Decision Points
- Measurement of maintenance difficulty

Design Metrics

Design goes for structure, therefore design metrics try to measure structure. Structure may be represented by graphs

- Control flow between statements
- Usage between modules
- Calls between modules (functions)
- Read/write between modules (variables)
- Inheritance between classes

→ typically, design metrics evaluate graphs

Measuring Graphs

- Graph attributes are
  - number of nodes n and edges e
  - depth and width of the graph
- Calculations of graphs
  - relationship between nodes and edges

$$m(G) = \frac{n}{e}$$

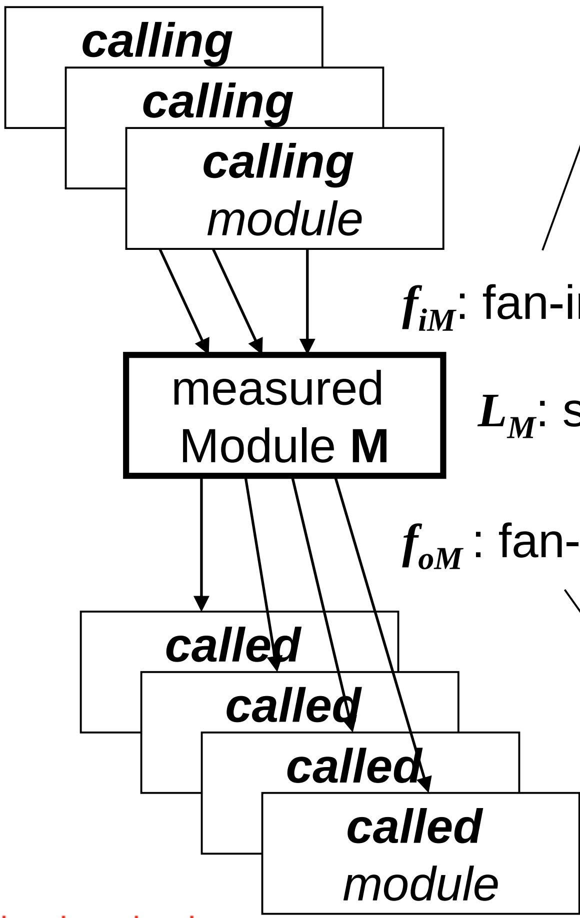
  - Tree derivation (0, if G is a tree; maximum, if G is a complete graph)

$$m(G) = \frac{2(e - n + 1)}{(n - 1)(n - 2)}$$

  - Cyclomatic number

$$V(G) = e - n + p;\ p = 2$$

Design Metrics – fan in/fan out



Workload, delegated **to** M

f_{iM}: fan-in of module M

L_M: size of module M

f_{oM} : fan-out of module M

Workload, delegated **by** M

Work, **done** by M

- Design complexity of a Module
- unweighted
  $C_M = (f_{iM} * f_{oM})^2$
- weighted
  $EC_M = L_M * (f_{iM} * f_{oM})^2$

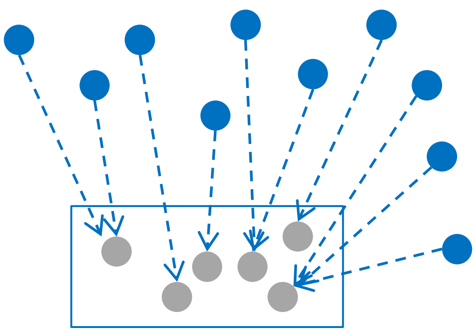
„Martin-Metrics“ – 1

Metrics defined by Robert C. Martin (Agile Software Development: Principles, Patterns, and Practices, Prentice Hall, 2002)

- **Number of Classes and Interfaces**
  - indicator of the extensibility of the package.
- **Abstractness (A)** of a package
  - The ratio of the number of abstract classes (and interfaces) in the analyzed package to the total number of classes in the analyzed package.
  - A=0 $\rightarrow$ a completely concrete package
  - A=1 $\rightarrow$ a completely abstract package.

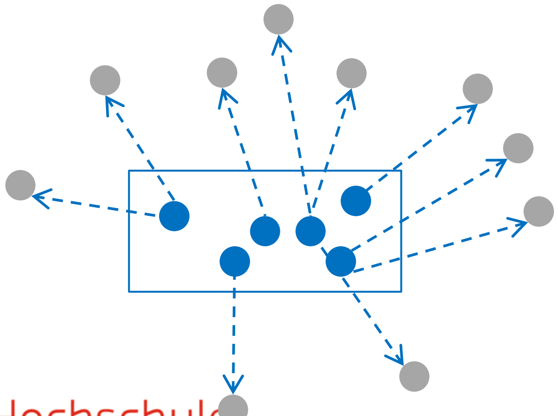
From: http://en.wikipedia.org/wiki/Software_package_metrics and http://web.cs.wpi.edu/~gpollice/Maps/CS4233/Week5/CS423 Lecture 15/index.html

„Martin-Metrics“ – 2



Afferent Coupling (CA) of a package

- the number of classes outside this package that depend on classes in this package.
- Indicator for the responsibility of the package



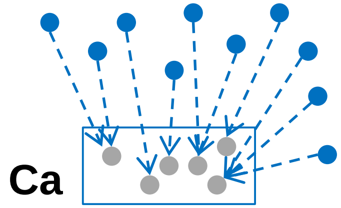
Efferent Coupling (CE) of a package

- the number of classes within this package that depend on classes outside the package.
- Indicator for the independence of the package

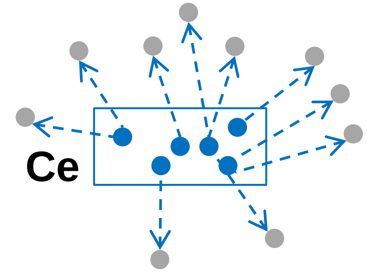
From: http://en.wikipedia.org/wiki/Software_package_metrics and http://web.cs.wpi.edu/~gpollice/Maps/CS4233/Week5/CS423 Lecture 15/index.html

„Martin-Metrics“ – 3

- Instability (I): $I = \frac{Ce}{(Ce+Ca)}$
  - indicator of the package's resilience to change
  - I=0 → a completely stable package
    (i.e. many incoming / few outgoing dependencies
    → difficult to modify)
  - I=1 → a completely instable package
    (i.e. many outgoing / few incoming dependencies
    → possibility of easy changes)



→ responsibility



→ independence

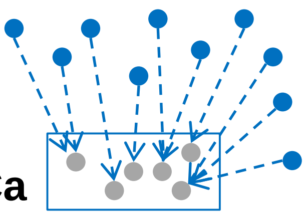
- Package Dependency Cycles
  - Dependency cycles should be ommitted

From: http://en.wikipedia.org/wiki/Software_package_metrics and
http://web.cs.wpi.edu/~gpollice/Maps/CS4233/Week5/CS423 Lecture 15/index.html

„Martin-Metrics“ – 4

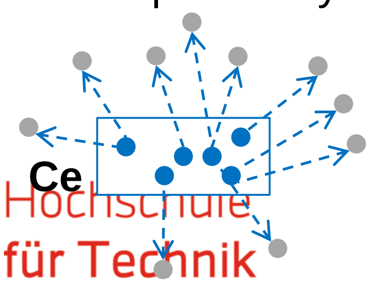
Abstractness A : $\frac{abstr\ Cl. + Interf.}{all\ Classes}$

Instability (I): $\frac{Ce}{(Ce + Ca)}$



Ca

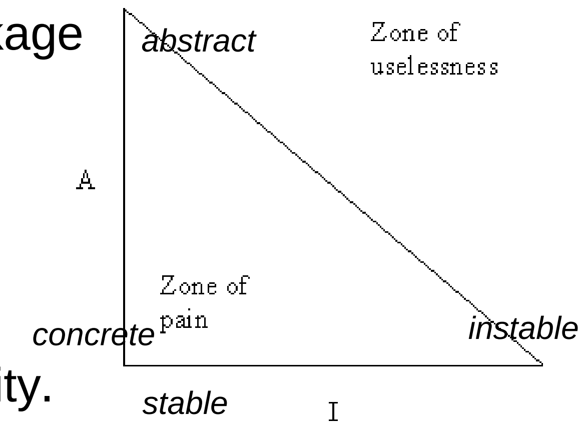
→ responsibility



Ce

→ independence

- Distance from the Main Sequence: D = | A + I - 1 |
  - The perpendicular distance of a package from the idealized line A + I = 1.
  - indicator of the package's balance between abstractness and stability.
  - A package squarely on the main sequence is optimally balanced with respect to its abstractness and stability.
  - Ideal packages are either completely stable and abstract (x=0, y=1)
  - or completely instable and concrete (x=1, y=0).
  - D=0 → a package that is coincident with the main sequence
  - D=1 → a package that is as far from the main sequence as possible.



From: http://en.wikipedia.org/wiki/Software_package_metrics and http://web.cs.wpi.edu/~gpollice/Maps/CS4233/Week5/CS423 Lecture 15/index.html

Specification Metrics

There are several ways to measure “specification”

- Count elements: pages, use-cases, user-stories
- Count “Functionality”:
  - Number of specified business functions / actors,
  - Volume of user-/DB-interaction (→ Function Point Measurement)

Metrics Examples for Object-Oriented Systems

Structure	Measurement / Number
Package	■ Number of Classes within a Package – NoClassesP ■ efferent Coupling Between Packages – effCBP ■ afferent Coupling Between Packages – affCBP
Class	■ Number of Parents – NoP ■ efferent Coupling Between Objects – effCBO ■ afferent Coupling Between Objects – affCBO ■ Number of Public Attributes – NoPubA ■ Number of Public Methods – NoPubM ■ Lack of Cohesion of Methods – LCOM ■ Weighted Methods Count – WMC (see below) ■ Number of directly Inherited Methods in a class – N ■ Number of Directly Overridden Methods in a class – ■ Number of Added Methods in a class – NoAddM
Attribute	■ Number of Get and Set Methods – NoGetSetM
Method	■ Number of Characters of Code – CoC ■ Lines of Code – LOC ■ Cyclomatic Complexity - CC

Number of classes within this package that depend on classes outside this package. Indicator for the independence of the package.

Number of classes outside of this package that depend on classes within this package. Indicator of the responsibility of the package.

for example: Number of independent groups of variables (i.e. addressed by methods); n > 1 → not coherent

Also for applicable on imperative programs

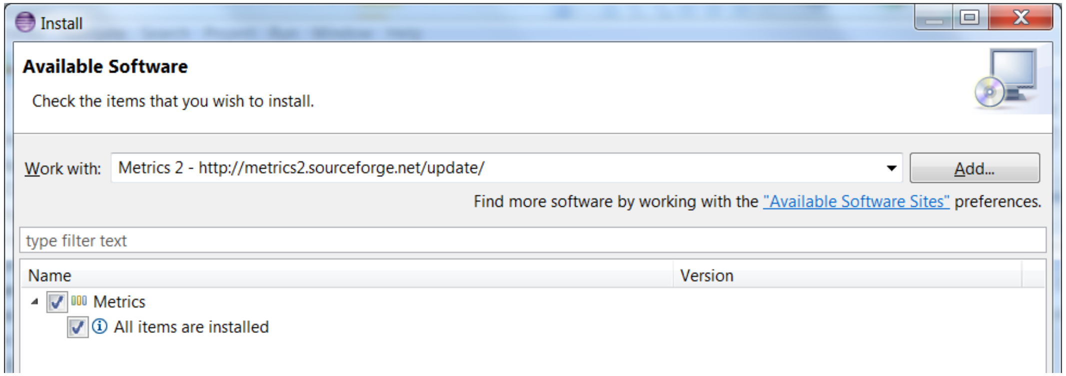
Hochschule
für Technik
Stuttgart

Metric-Tools

- Eclipse Metrics plugin (free http://metrics2.sourceforge.net/)
  - Measuring complexity
  - Visualization of dependencies
- McCabe – IQ2-Suite
  - Calculation and visualization of complexity metrics
  - Support of re-enegineering activities, cross references, call graphs, inheritance trees, dynamic analysis
- Crocodile – BTU Cottbus
  - Collection of data, analysis through metrics
  - Definition of new metrics in a definition language
  - Large visualization component
- Sotograph – Software Tomography
  - Analysis of subsystems and architectures
  - Visualization von layers, usage, …

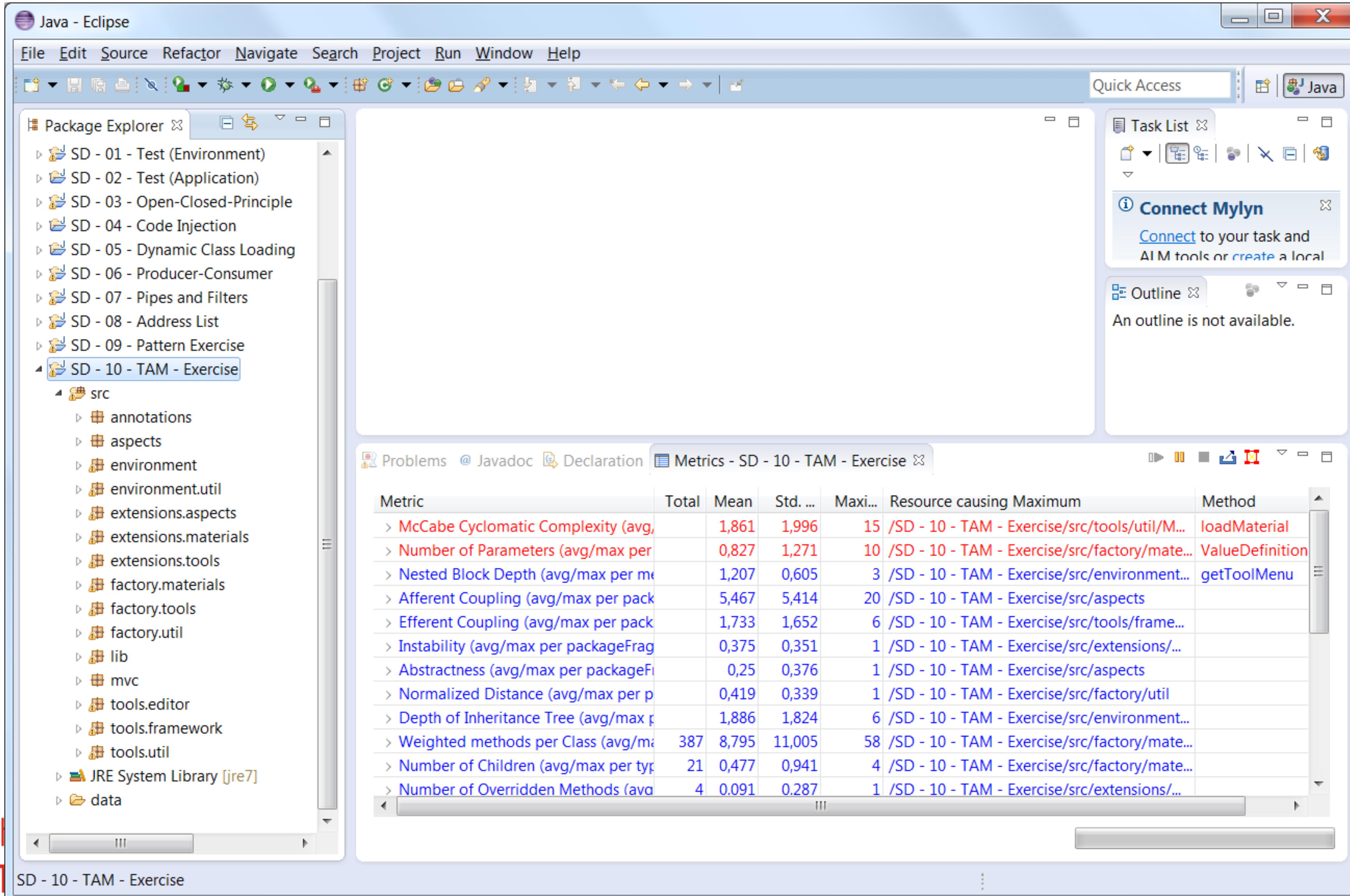
Example: Eclipse metrics

- Documentation: http://metrics2.sourceforge.net/
- Installation in Eclipse
  - Help> Install New Software ...
  - Repository Location: http://metrics2.sourceforge.net/update/



- Usage
  - Select project
  - Properties > metrics > ☒
  - Window > Show View> Metrics View

Example: Eclipse metrics



Bonus: Industrial Software Metrics Top 10 List (1)

B. W. Boehm: Industrial software metrics top 10 list. IEEE Software, September (1987), 84-85.

1. Finding and fixing a software problem after delivery is 100 times more expensive than finding and fixing it during the requirements and early design phases.
2. You can compress a software development schedule up to 25% of nominal, but no more.
3. For every dollar you spend on software development you will spend two dollars on software maintenance.
4. Software development and maintenance costs are primarily a function of the number of source instructions in the product.
5. Variations between people account for the biggest differences in software productivity.
6. The overall ratio of computer software to hardware costs has gone from 15:85 in 1955 to 85:15 in 1985, and it is still growing.
7. Only about 15% of software product-development effort is devoted to programming.

Bonus: Industrial Software Metrics Top 10 List (2)

B. W. Boehm: Industrial software metrics top 10 list. IEEE Software, September (1987), 84-85.

8. Software systems and software products each typically cost three times as much per instruction to fully develop as does an individual software program. Software-system products cost nine times as much.
9. Walkthroughs catch 60% of the errors.
10. Many software phenomena follow a Pareto distribution: 80% of the contribution comes from 20% of the contributors. Some examples:
    - 20 percent of the modules contribute 80 percent of the errors (not necessarily the same ones),
    - 20 percent of the errors consume 80 percent of the cost to fix,
    - 20 percent of the modules consume 80 percent of the execution time, and
    - 20 percent of the tools experience 80 percent of the tool usage.