EDAA35 – Lecture 3
Software Metrics

Martin Höst

Feb. 6, 2020

Outline

1. Introduction
2. Some metrics theory
   - Ways of categorizing metrics
   - Scales
   - GQM
   - Metrics definition and validation
3. Quality factors
4. Example metrics
   - Size
   - Algorithmic cost estimation
   - Complexity and coupling
   - Instability
   - Industrial context

Course information

- Lab 2 this week (probably takes more than 2 h)
- Read both R chapters before the lab, and do the small preparation assignment
- Access to files from home (ssh, sftp, scp, etc):
  http://www.ddg.lth.se/perf/unix/unix-x.pdf
  > scp STIL@login.student.lth.se:/usr/local/cs/EDAA35/data.txt .
- Lab 2a hint: in a loop go through every column and add to a new data frame
- Lab 2b hints: choose right separator in read.csv and stringsAsFactors=F, add own column names, tapply probably useful
- Lecture 27 Feb in E:A instead

Software metrics

- "You cannot control what you cannot measure" (DeMarco)
- Project tracking – understanding where you are
- Understanding quality during development
- Understanding performance of different methods/tools/...
- As a basis for process improvement
- As a basis for empirical studies
- Understanding the quality of products

Some terms informally

- Metric – definition of how to measure the attribute of interest, i.e. $M(A)$
- Measurement – what you do in order to get a number for an attribute in the real world, i.e. $val = M(A)$
- a basic requirement on metrics is that they should *preserve the empirical observation*, i.e. if A is bigger than B in the real world then $M(A) > M(B)$

Application of metrics

1. Define / decide metrics to use
2. Validate metrics
3. Collect metrics
4. Analyze metrics
5. Draw conclusions

Example: Code inspection statistis

- Total KLOC inspected: 27
- Average preparation rate: 194 LOC/h
- Average inspection rate: 172 LOC/h
- Total faults detected per KLOC: 106

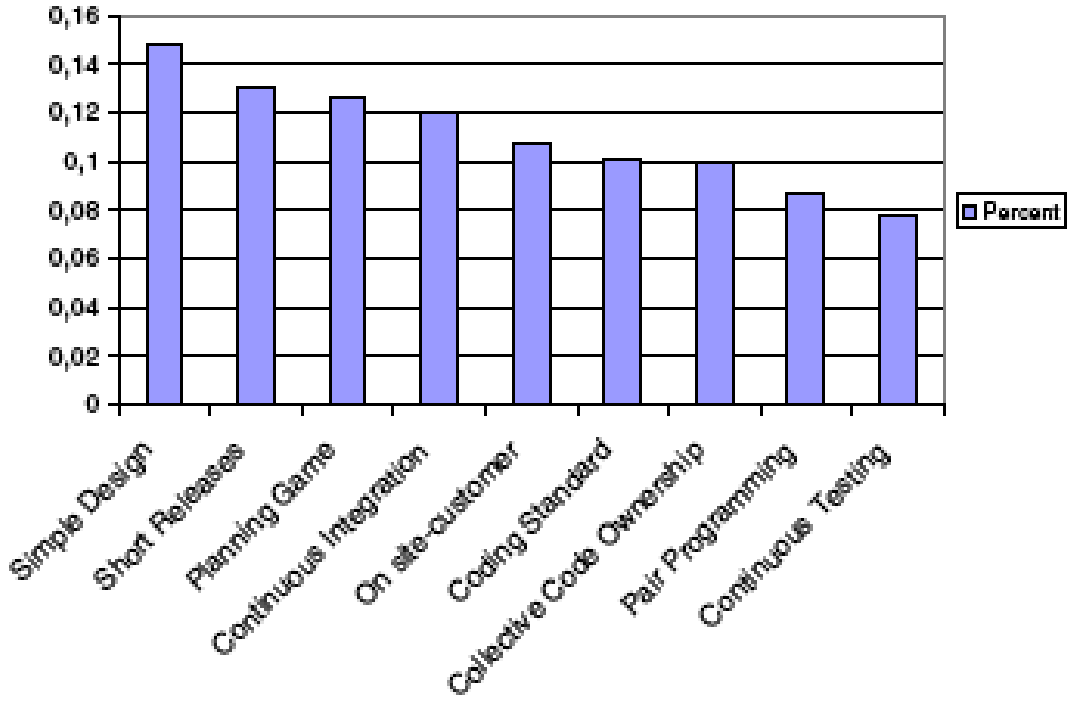
Fenton, Pfleeger, Software Metrics, 2:nd ed., PWS Publishing Company, 1997

Example: Size metrics from OSS project

Code Metric	2004v	2008v
LOC: lines of code	840,502	1,442,225
ELOC: effective LOC	650,055	1,110,261
C: # comment lines	484,349	630,635
TCC: total complexity	167,753	300,493
FFC: total number of file functions	15,588	45,216

Oručević-Alagić, Höst, A Case Study on the Transformation From Proprietary to Open Source Software, OSS, 2010

Example: Ease of introduction of XP practices



Svensson, Höst, Introducing an Agile Process in a Software Maintenance and Evolution Organization, CSMR, 2005

What to measure?

Products e.g. size of code documents, modularity of requirements specifications, complexity of design, coverage level of test data,...

Processes e.g. time of construction activities, number of faults found in test phase, effort in design phase,..

Resources e.g. price of software components, age of personnel, experinece of personnel,...

Internal / external attributes

Internal attributes of a product, process, or resource are those which can be measured purely in terms of the product, process, or resource itself. *Examples*: size, modularity, age, price,...

External attributes of a product, process, or resource are those which can only be measured with respect to how the product, process, or resource relates to its environment. *Examples*: comprehensability, reliability, cost-effectiveness, productivity, experience,...

Often you measure internal attributes when you would like to measure external.

Objective or subjective?

Objective e.g. size of code documents measured with unix "wc", time of construction activities taken from time reporting system, complexity of modules measured with "cyclomatic compexity"...

Subjective e.g. grading of modules wrt complexity on Likert scale, ranking of third party components wrt understandability of API,...

Different scales

- Nominal scale
- Ordinal scale
- Interval scale
- Ratio scale

Different scales – some examples

- Nominal scale: type of fault (data, function, interface,...)
- Ordinal scale: criticality of fault (class A, B, C, etc)
- Interval scale:
- Ratio scale: number of faults, debugging time,...

Different scales – defining relations

- Nominal scale: $=$
- Ordinal scale: $=$, $>$
- Interval scale: $=$, $>$, ratio of intervals
- Ratio scale: $=$, $>$, ratio of intervals, ratio of values

Different scales – "allowed" transformations

- Nominal scale: labeling, classifying entities
- Ordinal scale: $M(x) > M(y) \rightarrow M'(x) > M'(y)$
- Interval scale: $M' = aM + b$
- Ratio scale: $M' = aM$

Different scales – examples of "allowed" statistics

- Nominal scale: mode, frequency
- Ordinal scale: mode, frequency, median, correlation (Spearman)
- Interval scale: mean, standard deviation, correlation (Pearson)
- Ratio scale: median, mean, standard deviation, geometric mean $(\prod_{i=1}^{n} a_i)^{1/n}$

Fenton, Pfleeger, 1997

Metrics challenges

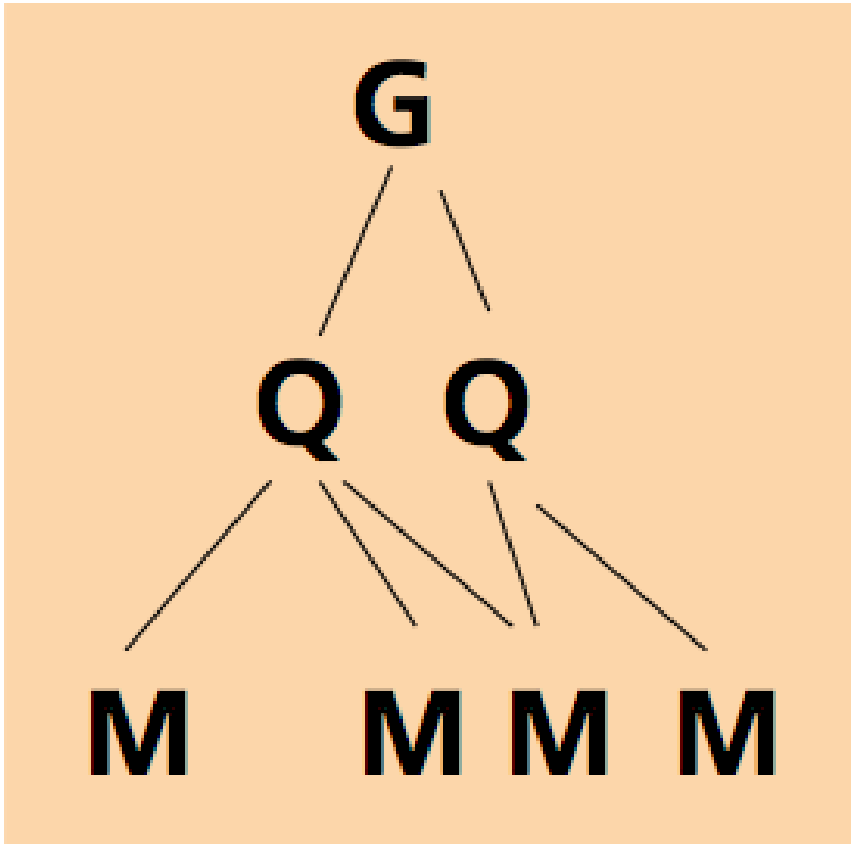
- Data collected in different projects should be comparable
- Data collected should be reliable and of high quality
- People do not want to collect data that is not used

$\Rightarrow$

- Avoid "data cemetaries"
- Collect only the data that you really need (but *all* data that you need)
- Understand and record the context in which data is collected
  - Product
  - Team
  - Process
  - ...
- Quality assurance for data collection

Goal Question Metrics (GQM)

- Goals: What is the organisation trying to achieve? The objective of process improvement is to satisfy these goals
- Questions: Questions about areas of uncertainty related to the goals. You need process knowledge to derive these
- Metrics: Measurements to be collected to answer the questions



GQM goal template

Analyze *object(s) of study*
for the purpose of *purpose*
with respect to their *quality focus*
from the point of view of *the perspective*
in the context of *context*

GQM goal examples

Object(s) of study the product or process investigated, e.g. test phase, inspection process, requirements document,...

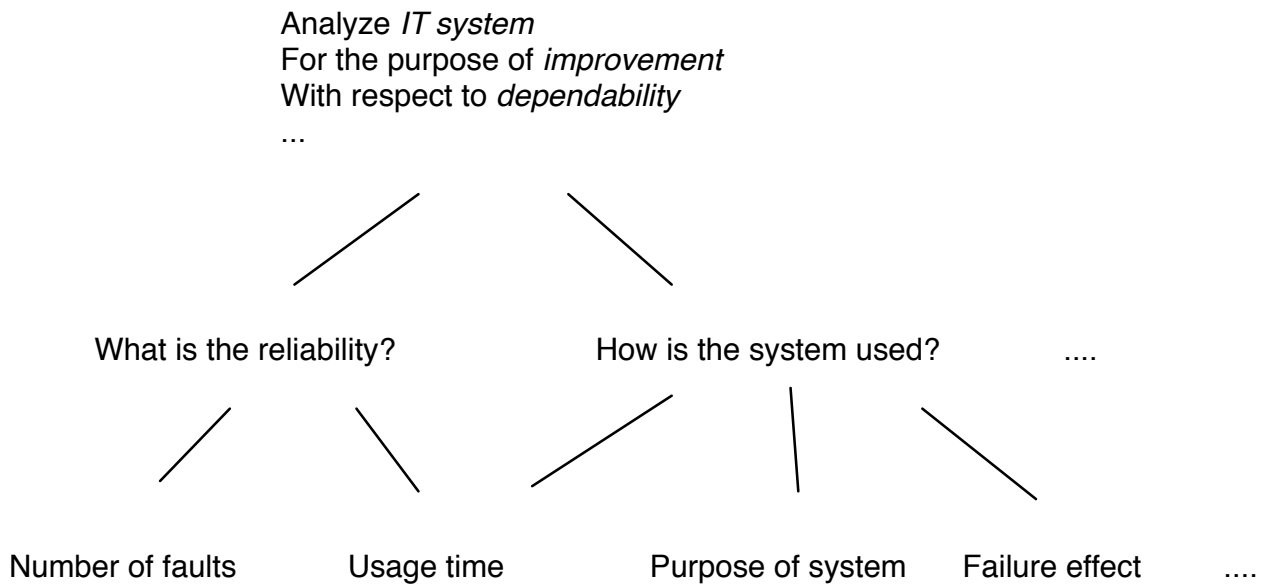
Purpose improvement, understanding,...

Quality focus i.e. reliability, maintainability, cost,...

Perspective customer, project leader, senior management,...

Context project, organization,...

Example (simplified)



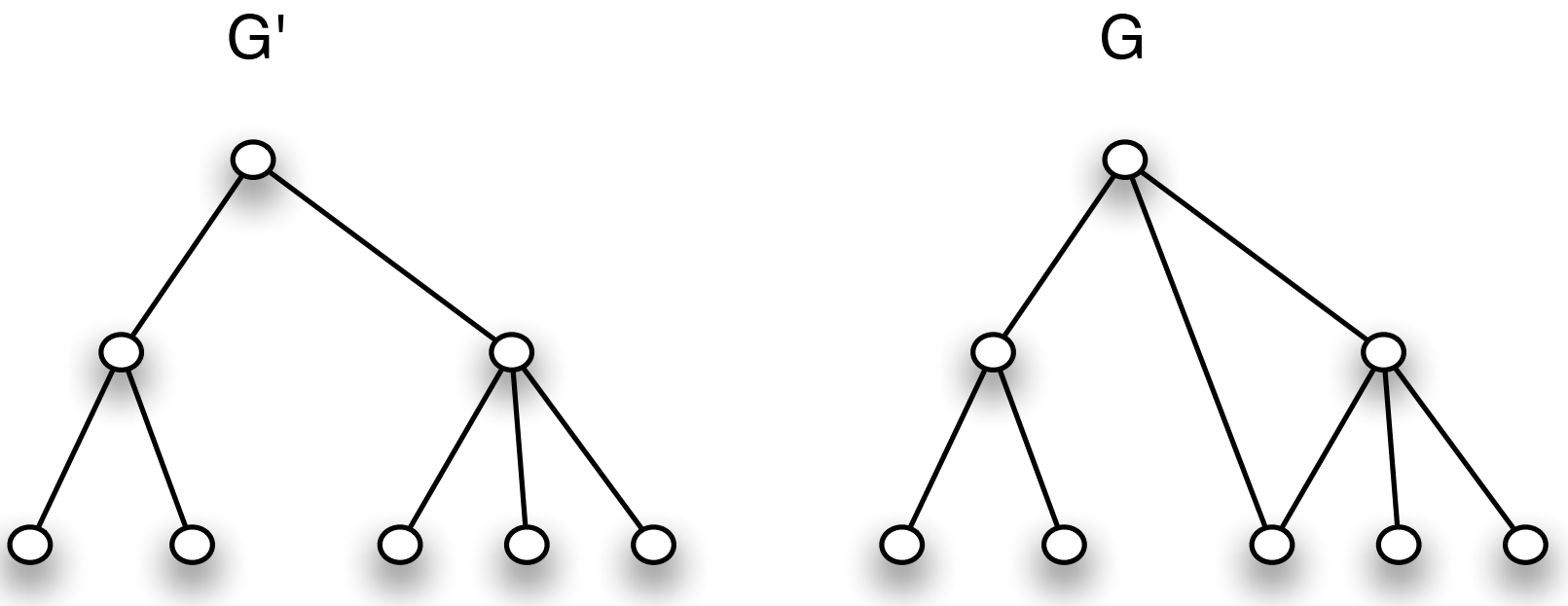
GQM abstraction sheet (an example of support)

Quality focus	Variation factors
1. Number detected failures 2. ...	1. Quality of test cases 2. ...
Baseline hypothesis	**Variation Hypothesis**
1. 30 2. ... (here it is wise to use distributions)	1. The higher the quality of the test cases, the more failures detected 2. ...

Metrics definition and validation

1. Check definitions, scales, etc
2. Compare definition of metric to understanding of phenomena
3. Compare collected empirical data to understanding of phenomena

Example: tree impurity of designs



Example: formulate understanding of phenomena

1. $m(G) = 0$ if and only if G is a tree
2. $m(G) > m(G')$ if G differs from G' only by the insertion of an extra edge
3. Let N denote #nodes and E #edges. If $N > N'$ and $E - N + 1 = E' - N' + 1$ then $m(G) < m(G')$ [1]
4. $m(G) \leq m(K_N)$ where N is the number of nodes in G and K_N is a complete graph with N nodes

[1] Number of "extra edges" is the same but G is larger than G'

Example: suggest metric

$$m(G) = \frac{\text{Number of extra edges}}{\text{Max number of extra edges}}$$

1. $m(G) = 0$ if and only if G is a tree:

$$m(G) = \frac{0}{\text{Max number of extra edges}} = 0$$

2. $m(G) > m(G')$ if G differs from G' only by the insertion of an extra edge:

$$\frac{K+1}{\text{Max number of extra edges}} > \frac{K}{\text{Max number of extra edges}}$$

$$K+1 > K$$

3. and 4. in the same way...

Quality factors

- There are a number of lists of quality requirements
  - McCall
  - ISO 9126
  - IEEE 830
  - ...

ISO 9126 (selected parts)

- Functionality
  - Accuracy, Security, Interoperability, Suitability, Compliance
- Reliability
  - Maturity, Fault tolerance, Recoverability
- Usability
  - Understandability, Learnability, Attractiveness
- Efficiency
  - Time behaviour, Resource utilization
- Maintainability
  - Testability, Changeability, Analyzability
- Portability
  - Adaptability, Installability, Conformance

Size

- One of the most often used metrics
- Basis for cost estimation
- Also used as normalization for other metrics, e.g. reliability
- Often measured as lines of source code (LOC)

LOC – some things to think about

- Differences between different programming languages
- How to measure?
  - blank lines?
  - comment lines?
  - broken lines?
  - count "{", "}", etc?
  - ...
  - $\Rightarrow$
    - Coding standard can help
    - Count statements instead? (or ";"?)
- What to measure: new, modified, deleted, reused,...

A small assignment

- Assume you were given a piece of software (a word processor) that was being developed and currently was in the acceptance test phase,
- and you were given the task to measure "reliability" in terms of time between failures.
- What would you measure?
- How would you measure?

A small assignment 2

- Assume you would like to measure how difficult a piece of code is to understand.
- What would you measure?
- How would you measure?

Algorithmic methods

Basic idea:

Estimations based on historical data in the form of measurements from earlier projects typically in the form of effort $= c \times \text{size}^k$

- gives an "objective" estimate
- a good experience base *and* a good size estimate gives a good estimate
- ...but not easy to find a good experience base...
- ...and not easy to get a good size estimate...

The original COCOMO model (COCOMO81)

Formula

$$\text{Effort} = c \times \text{Size}^k$$

Constants

System type	c	k
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

...derived by looking at a lot of old projects

Deprecated now, but the basic idea is still valid

An early size measure: Albrecht function points (FP)

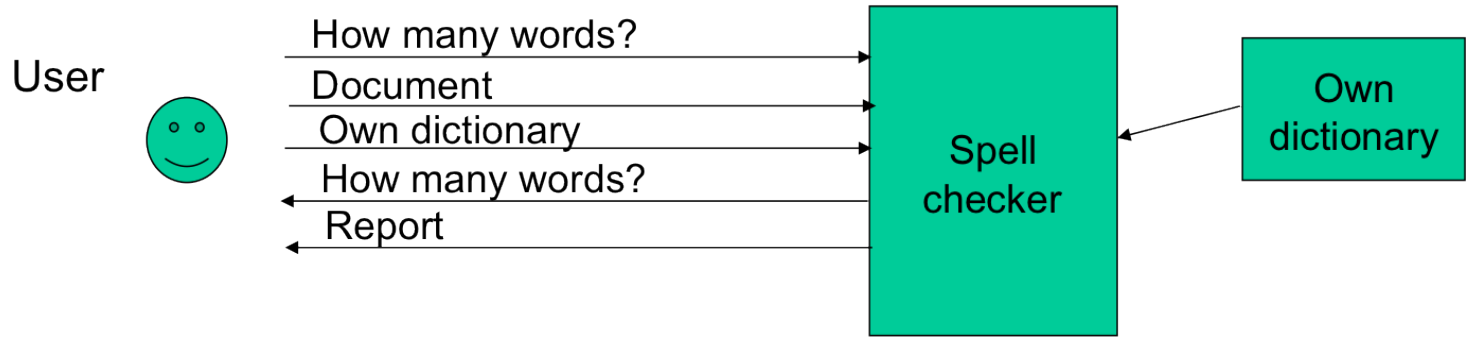
	Low	Medium	High
External input type	3	4	6
External output type	4	5	7
Logical internal file type	7	10	15
External interface file type	5	7	10
External inquiry type	3	4	6

- UFP $= \sum_{i=1}^{15}$(Number of items of weight $i \times$ weight$_i$)
- FP = UFP $\times$ "technical complexity factor"

UFP to SLOC Conversion Ratios

Language	Ratio (SLOC / UFP)
Assembly	320
C	128
C++	55
Java	53
PERL	27
Prolog	64

from COCOMO II Model Definition Manual



- Assume all weights are "medium"
- $A = \#$ ext input $= 2$ (doc, own dict)
- $B = \#$ ext out $= 2$ (report, # words)
- $C = \#$ queries $= 1$ (# words?)
- $D = \#$ ext files $= 2$ (doc, own dict)
- $E = \#$ int files $= 1$ (dict)
- $FP = 4A + 5B + 4C + 7D + 10E = 37$
- 37FP $\Rightarrow 37 \times 53 \approx 2000$ LOC (Java)

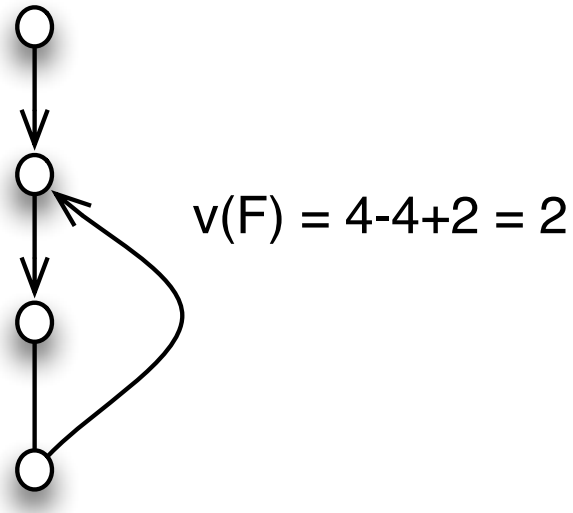
Complexity

- We want to measure the complexity of a program in a relevant way...
- E.g. in estimation and evaluation of maintainability
- Several attempts have been made
  - based on the code
  - based on the design

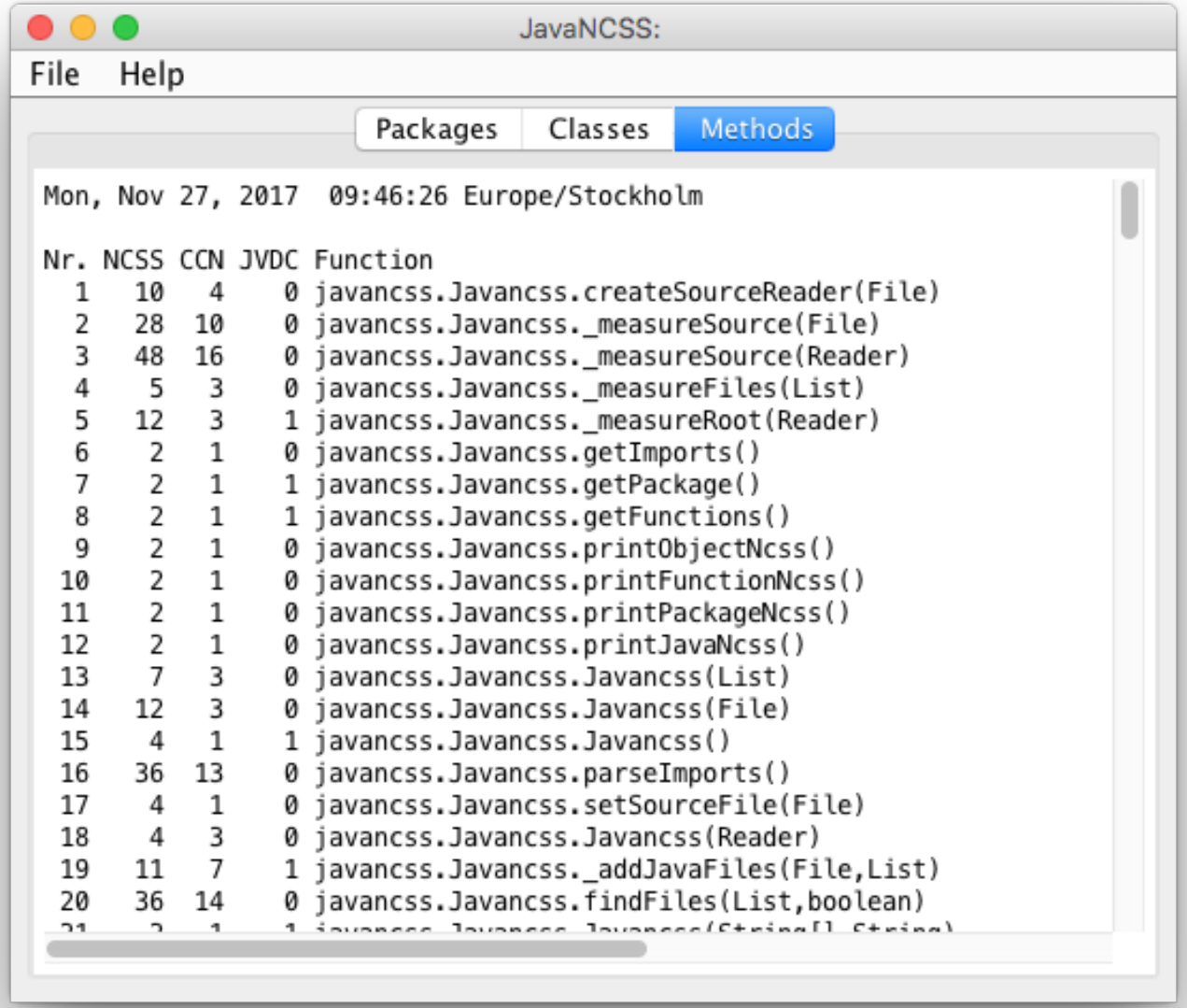
McCabe's cyclomatic complexity

- Represent the program as a flow graph
  - statements as nodes (n nodes)
  - flows as edges (e edges)
- Calculate cyclomatic complexity as the the number of *linearly independent paths*
  $v(F) = e - n + 2$

```
a <- 1:10
while (length(a) > 5) {
    print(a)
    a <- a[1:(length(a)-1)]
}
```



Tool example: JavaNCSS



Coupling and cohesion

- Coupling: how coupled two modules are – you don't want too much
- Cohesion: how cohesive one module is – that is good, you want that

Coupling (design)

x and y two modules. Example ordinal measure of coupling:

1. x and y have no communication
2. x and y communicate by simple parameters
3. x and y use the same record type as parameter
4. x passes a parameter ty y with the intention of controlling its behavior
5. x and y refer to the same global data
6. x refers to the inside of y

Fenton, Pfleeger, 1997

Cohesion (design)

1. Coincidental
2. Logical, e.g., "if A then function 1 else function 2")
3. Temporal, e.g., "read data", "analyse", "present result"
4. Procedurally, i.e. as temporal with related action
5. Communicational, around one data set
6. Functional: all elements of one functionality in one module AND all elemnts in the module are related to that functionality
7. Informational, e.g. all attributes and methods strongly interdependent and essential to the object

Pfleeger, Atlee, Software Engineering, 4:h ed., Pearson, 2010

Object oriented metrics (Chidamber och Kemerer)

- WMC: weighted methods per class (weighted for complexity)
- DIT: depth of inheritance tree
- NOC: number of children (number of immediate successor)
- CBO: coupling between object classes (number of classes to which the class is coupled)
- RFC: response for class (number of local methods plus number of methods called by local methods, i.e. $RFC = |RS|$ where $RS = \{M\} \bigcup_{\forall i} \{R_i\}$)
- LCOM: lack of cohesion metric (based on number of disjoint sets of local methods)

Fenton, Pfleeger, 1997

Metrics of "instability" (Martin)

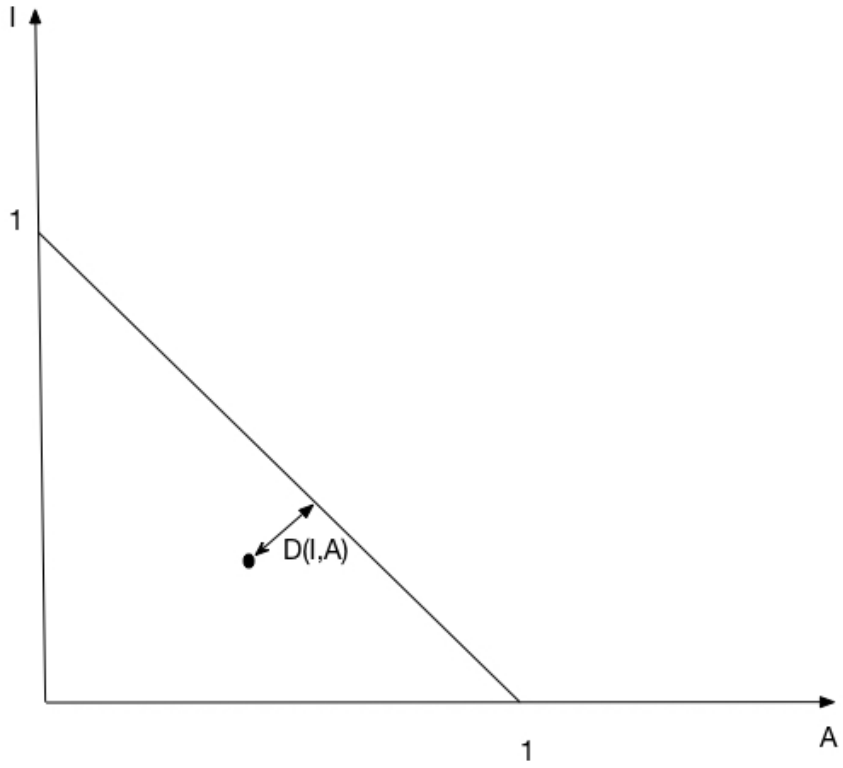
For one package:

- $C_e = \#$ classes depending on classes in other packages
- $C_a = \#$ classes in other packages depending on its classes
- $I = \frac{C_e}{C_a + C_e}$
- $I = 0$: maximum stability
  (no classes in package depend on classes in other packages, but classes outside package depending on classes in package)
- $I = 1$: maximum instability
  (no classes outside package depending on classes in package, but classes in packe depending on classes outside package)

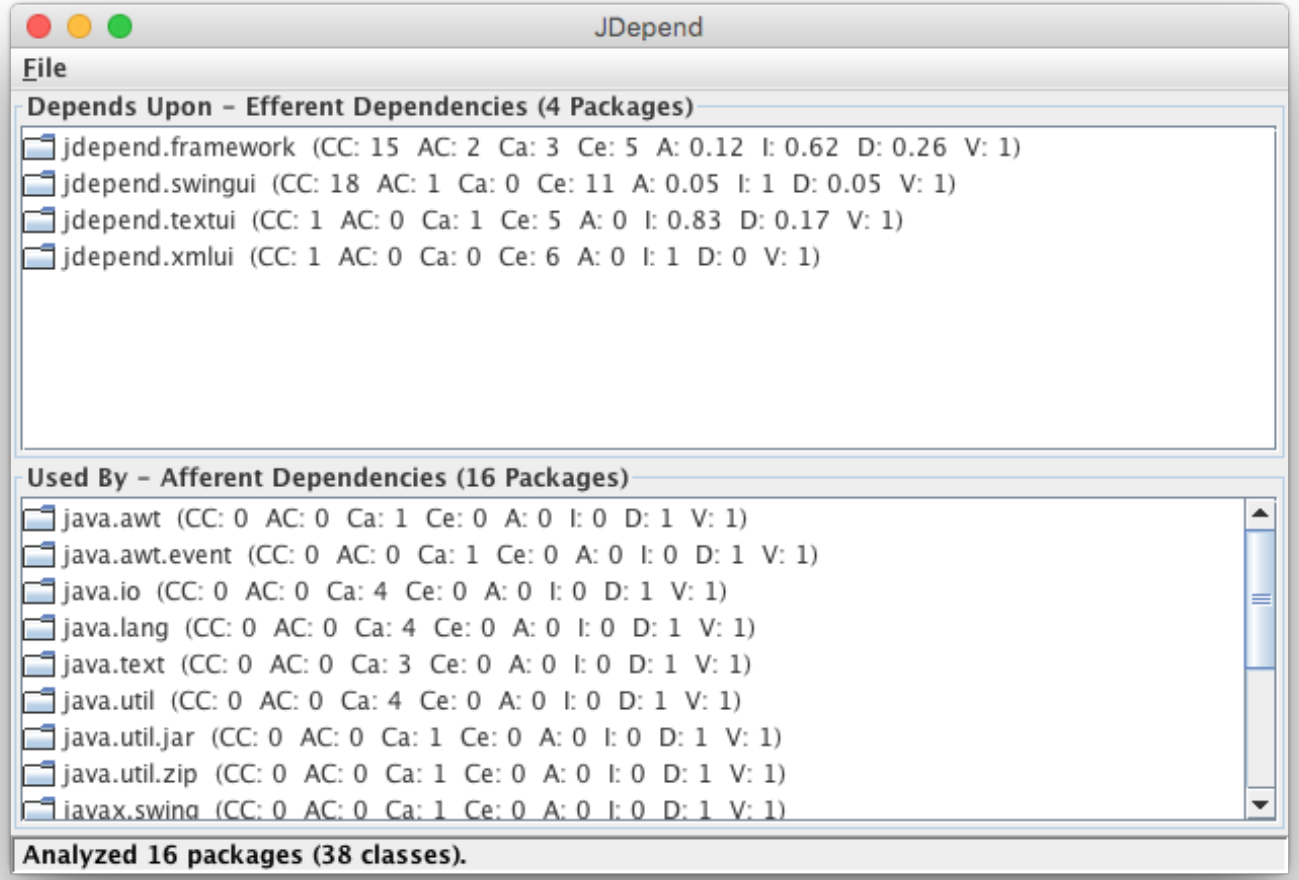
"Instability" cont.

For one package:

- $A = \frac{\#\text{ abstract classes}}{\#\text{ classes}}$
- "main sequence": $I + A = 1$



Tool example: JDepend



Examples of CMM questions (level 2) related to software metrics

- 1.1.3 – Does the SQA function have a management reporting channel separate from the software development project management?
- 2.1.14 – Is there a formal procedure used to make estimates of software size?
- 2.1.16 – Is there a formal procedure used to make estimates of software cost?
- 2.2.4 – Are statistics on software software code and test errors gathered?

Software process improvement