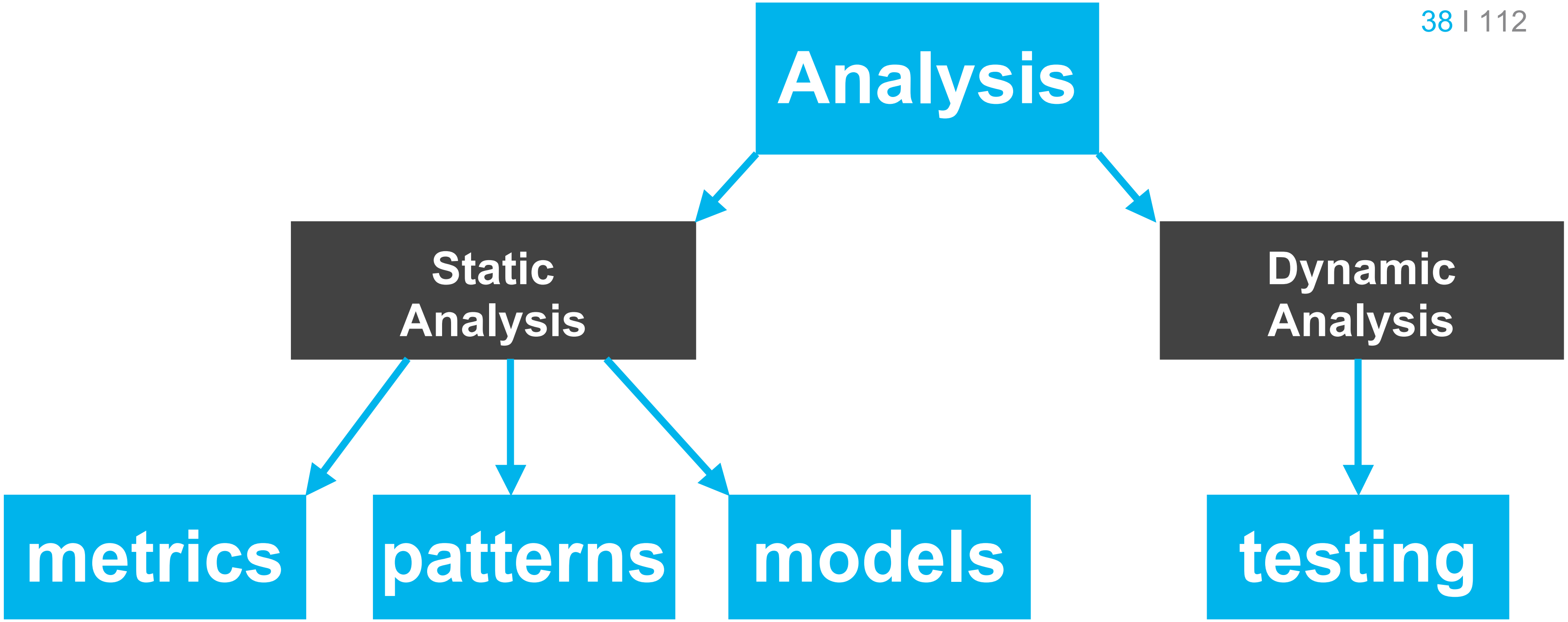
Structure of the lecture



PATTERNS

Patterns

Software Improvement Group

Coding style and coding standards

- E.g. layout, identifiers, method length, …

Secure coding guidelines

- E.g. SQL injection, stack trace visibility

Bug patterns

- E.g. null pointer dereferencing, bounds checking

Code smells

- E.g. “god class”, “greedy class”, ..

Patterns
Style and standards

Checking coding style and coding standards

- Layout rules (boring)
- Identifier conventions
- Length of methods
- Depth of conditionals

Aim

- Consistency across different developers
- Ensure maintainability

Tools

- E.g. CheckStyle, PMD, …
- Integrated into IDE, into nightly build
- Can be customized

Checking secure coding guidelines

- SQL injection attack
- Storing and sending passwords
- Stack-trace leaking
- Cross-site scripting

Aim

- Ensure security
- Security = Confidentiality + Integrity + Availability

Tools

- E.g. Fortify, Coverity

Detecting bug patterns

- Null-dereferencing
- Lack of array bounds checking
- Buffer overflow

Aim

- Correctness
- Compensate for weak type checks

Tools:

- e.g. FindBugs
- Esp. for C, C++

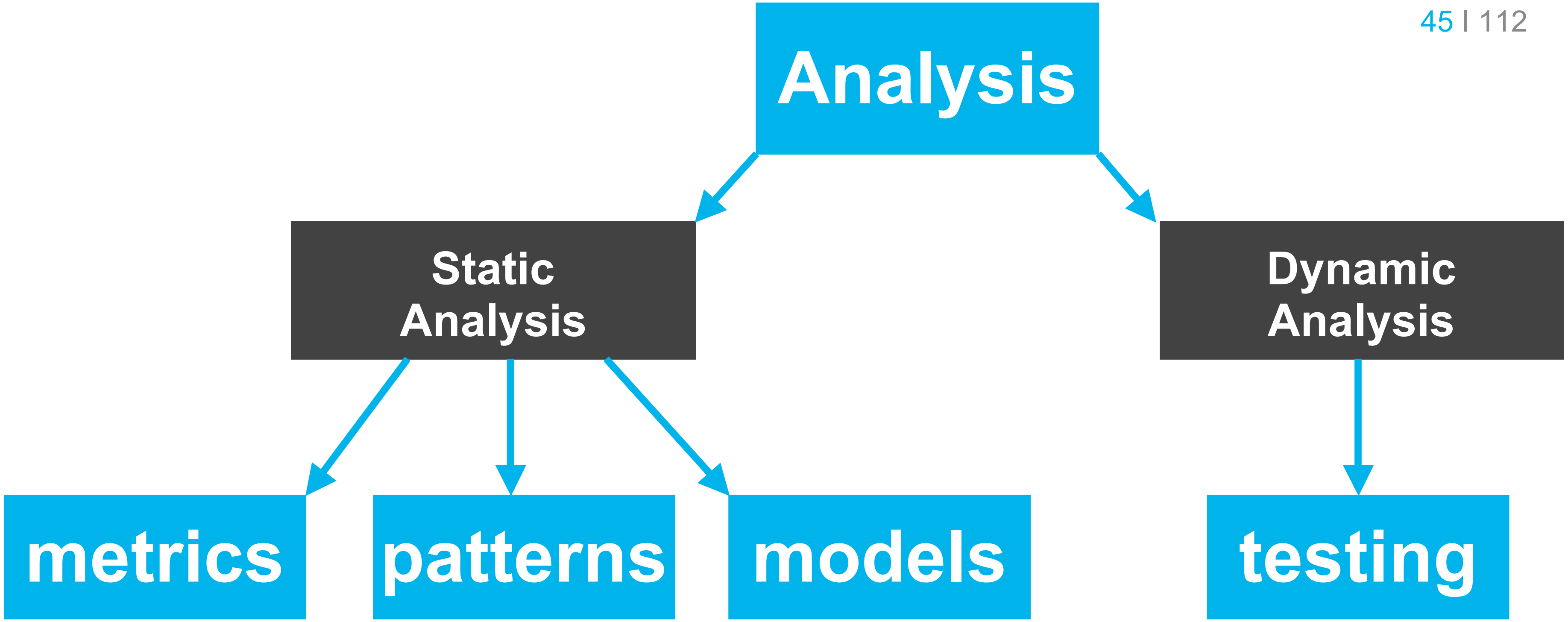
Run PMD / Checkstyle / FindBugs

- E.g. on a project of your own
- E.g. on some (easy-to-compile) open source project

Inspect results

- False or true positives?

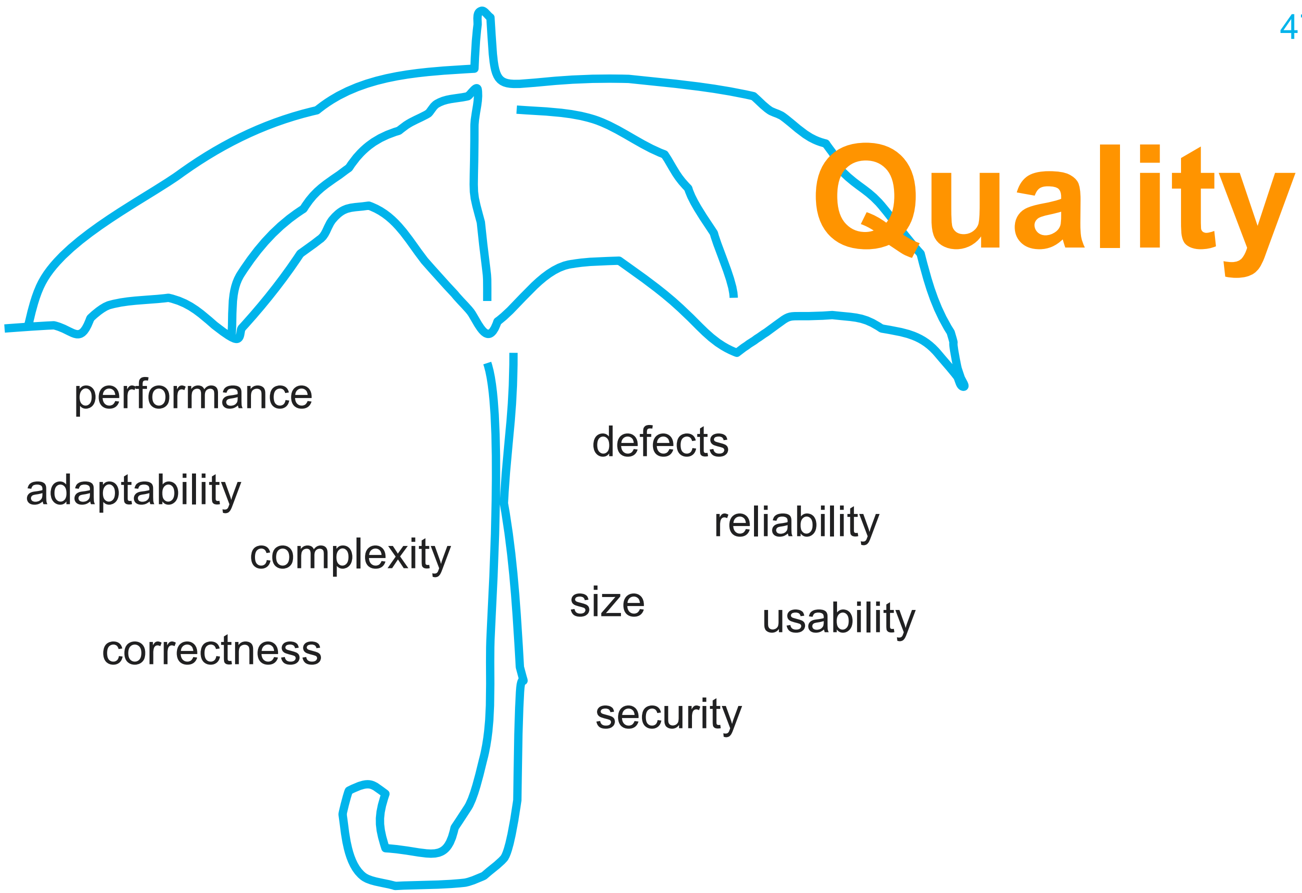
Structure of the lecture



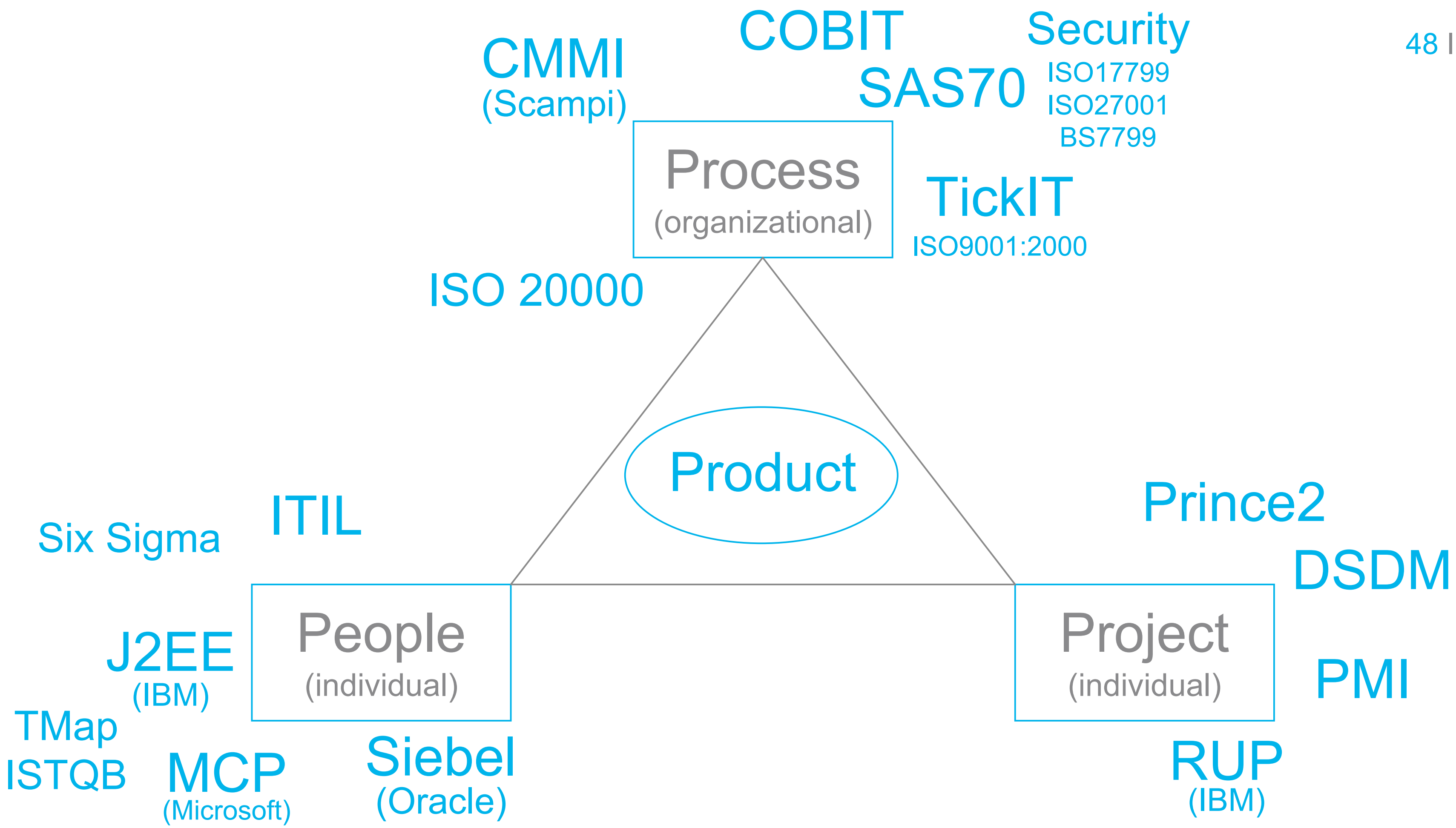
Software Improvement Group

METRICS & QUALITY

Software analysis
What?



The bermuda triangle of software quality



Software Improvement Group

Capability Maturity Model® Integration (CMMI®)

- "… is a process improvement approach that provides organizations with the essential elements of effective processes.." (SEI)
- CMMI for Development (CMMI-DEV), Version 1.2, August 2006.
- consists of 22 process areas with capability or maturity levels.
- CMMI was created and is maintained by a team consisting of members from industry, government, and the Software Engineering Institute (SEI)
- http://www.sei.cmu.edu/cmmi

The Standard CMMI Appraisal Method for Process Improvement (SCAMPI)

- "… is the official SEI method to provide benchmark-quality ratings relative to CMMI models."

Software Quality Process

Software Improvement Group

http://sas.sei.cmu.edu/pars/

Software Engineering Institute | Carnegie Mellon

Organization

Organization Name:	Accenture
Appraisal Sponsor Name:	Jack Ramsay, Marco Spaziani Testa, Maria Angeles Ramirez
Lead Appraiser Name:	John Voss
SEI Partner Name:	Accenture LLP

Model Scope and Appraisal Ratings

Level 2		Level 3		Level 4		Level 5	
Satisfied	REQM	Satisfied	RD	Out of Scope	OPP	Out of Scope	OID
Satisfied	PP	Satisfied	TS	Out of Scope	QPM	Out of Scope	CAR
Satisfied	PMC	Satisfied	PI				
Not Applicable	SAM	Satisfied	VER				
Satisfied	MA	Satisfied	VAL				
Satisfied	PPQA	Satisfied	OPF				
Satisfied	CM	Satisfied	OPD				
		Satisfied	OT				
		Satisfied	IPM				
		Satisfied	RSKM				
		Satisfied	DAR				

Organizational Unit Maturity Level Rating: 3

Additional Information for Appraisals Resulting in Capability or Maturity Level 4 or 5 Ratings:

Software Improvement Group

Levels

- L1: Initial
- L2: Managed
- L3: Defined
- L4: Quantitatively Managed
- L5: Optimizing

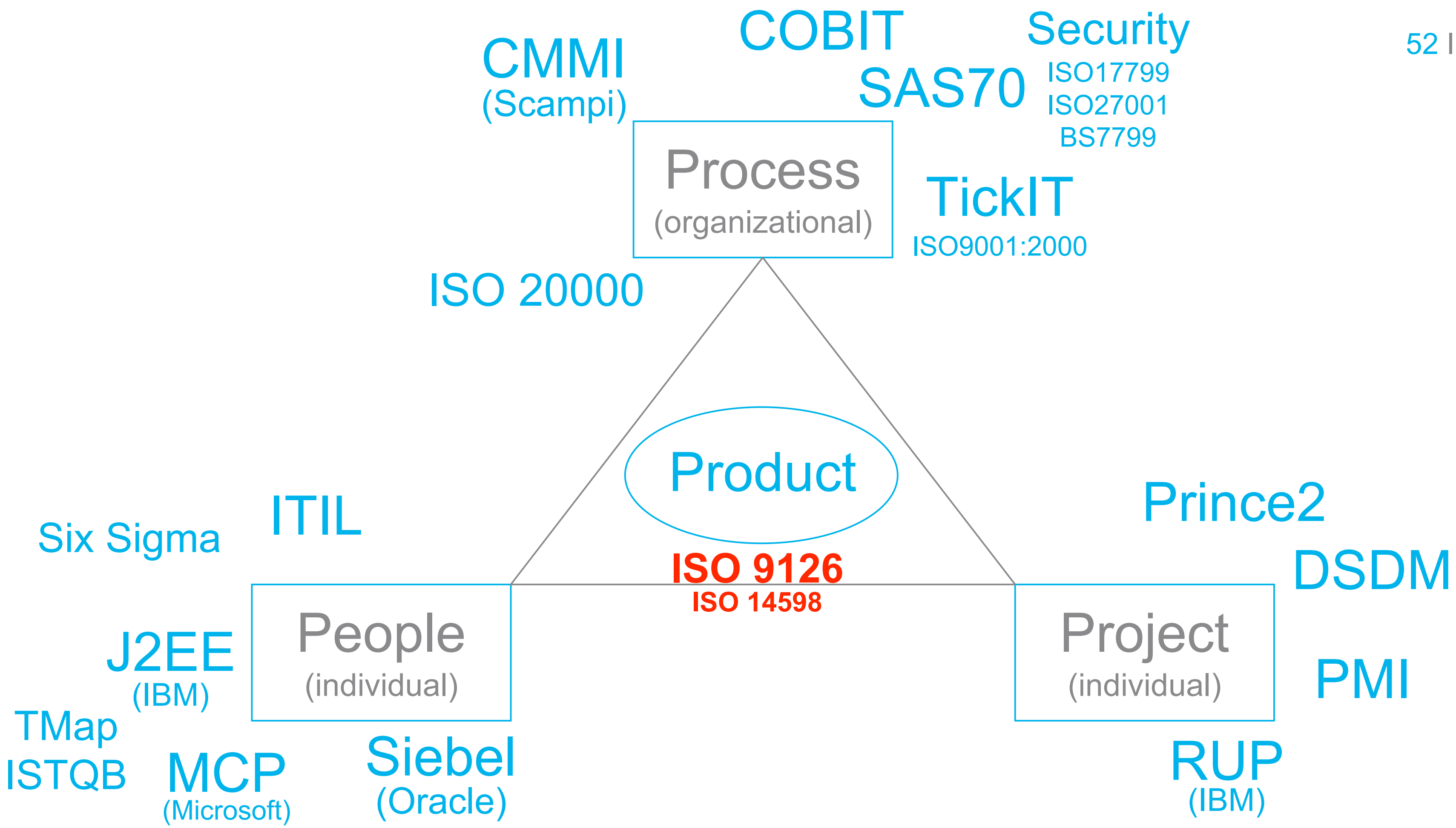
http://www.cmmi.de
(browser)

Process Areas

- Causal Analysis and Resolution
- Configuration Management
- Decision Analysis and Resolution
- Integrated Project Management
- Measurement and Analysis
- Organizational Innovation and Deployment
- Organizational Process Definition
- Organizational Process Focus
- Organizational Process Performance
- Organizational Training
- Product Integration
- Project Monitoring and Control
- CMMI Project Planning
- Process and Product Quality Assurance
- Quantitative Project Management
- Requirements Development
- Requirements Management
- Risk Management
- Supplier Agreement Management
- Technical Solution
- Validation
- Verification

CMMI®

The bermuda triangle of software quality



But …

What is software quality?

What are the technical and functional aspects of quality?

How can technical and functional quality be measured?

Software product quality standards

Software Improvement Group

ISO/IEC 9126

Software engineering -- Product quality

1. Quality model
2. External metrics
3. Internal metrics
4. Quality in use metrics

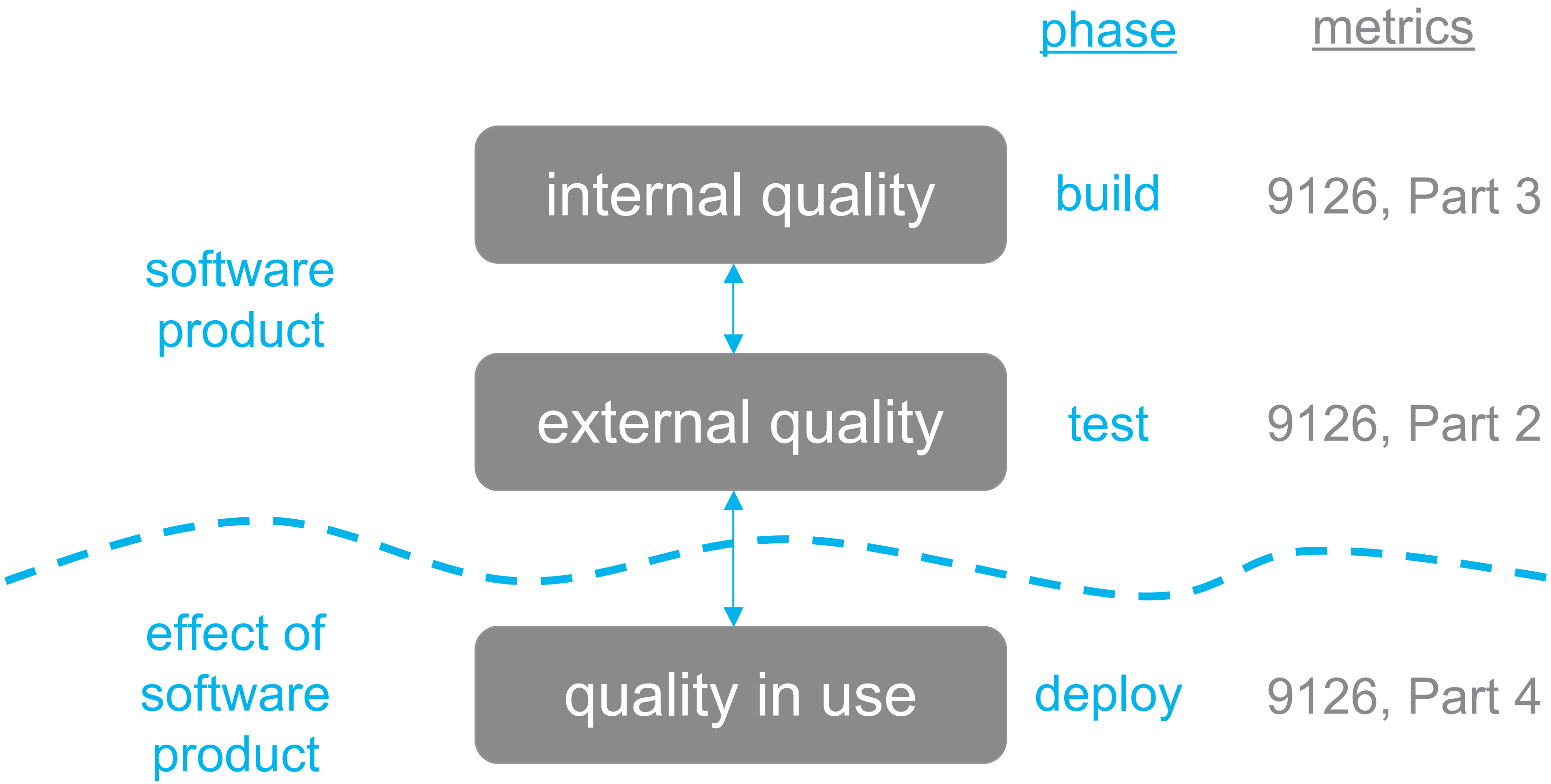
International Organization for Standardization

ISO/IEC 14598

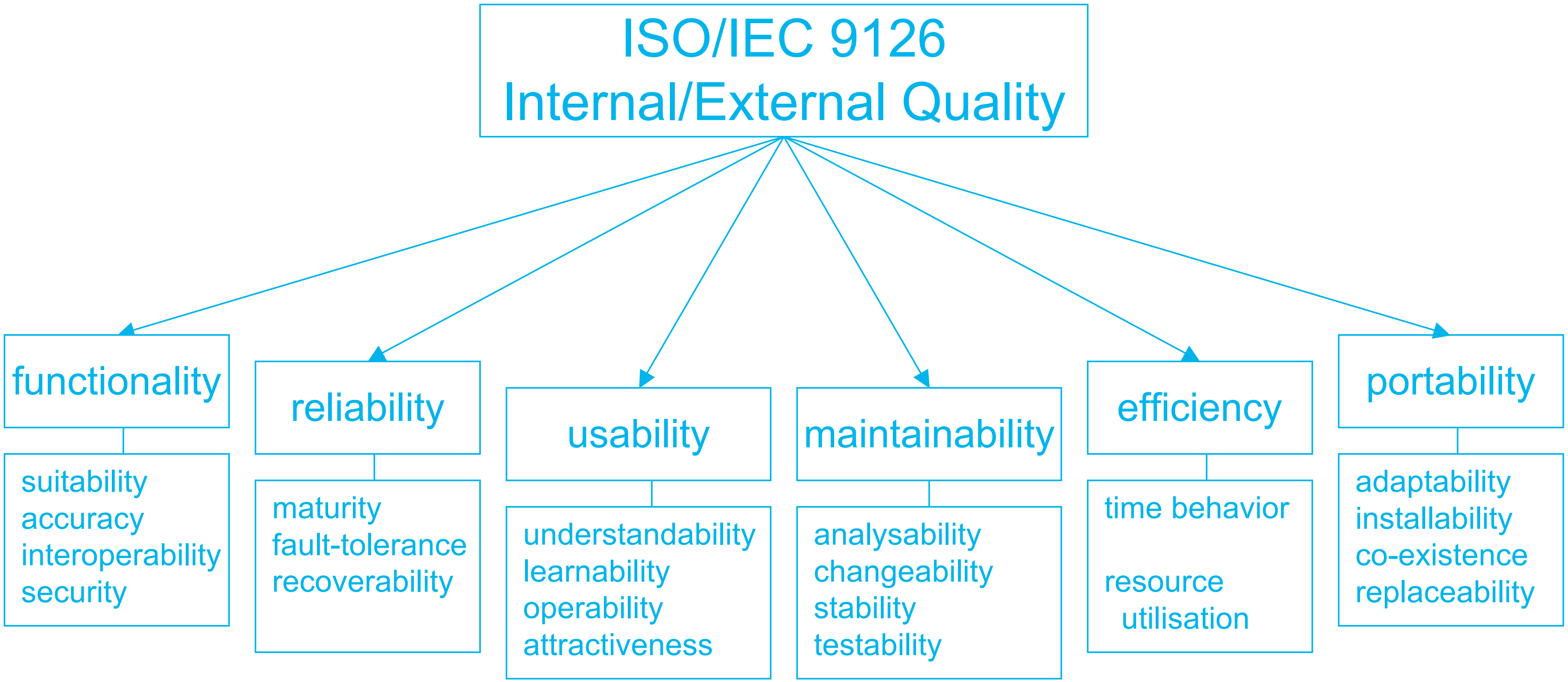
Information technology -- Software product evaluation

1. General overview
2. Planning and management
3. Process for developers
4. Process for acquirers
5. Process for evaluators
6. Documentation of evaluation modules

ISO/IEC 9126, Part 1
Quality perspectives



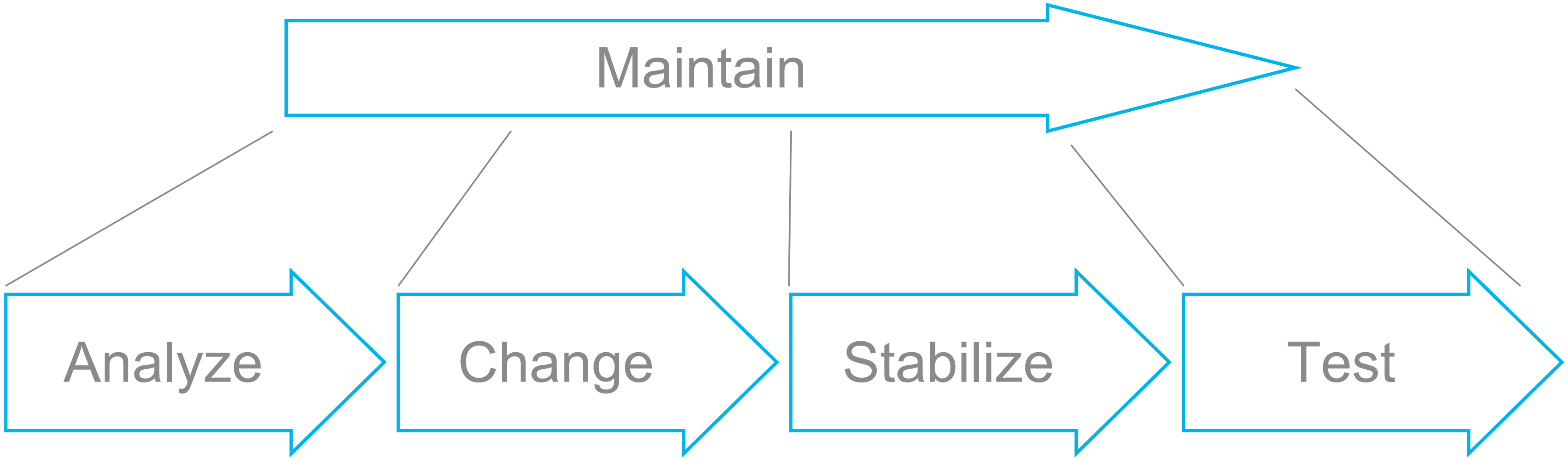
ISO/IEC 9126, Part 1
Product quality model: internal and external



ISO 9126, Part 1
Maintainability (= evolvability)

Maintainability =

- *Analyzability*: easy to understand where and how to modify?
- *Changeability*: easy to perform modification?
- *Stability*: easy to keep coherent when modifying?
- *Testability*: easy to test after modification?

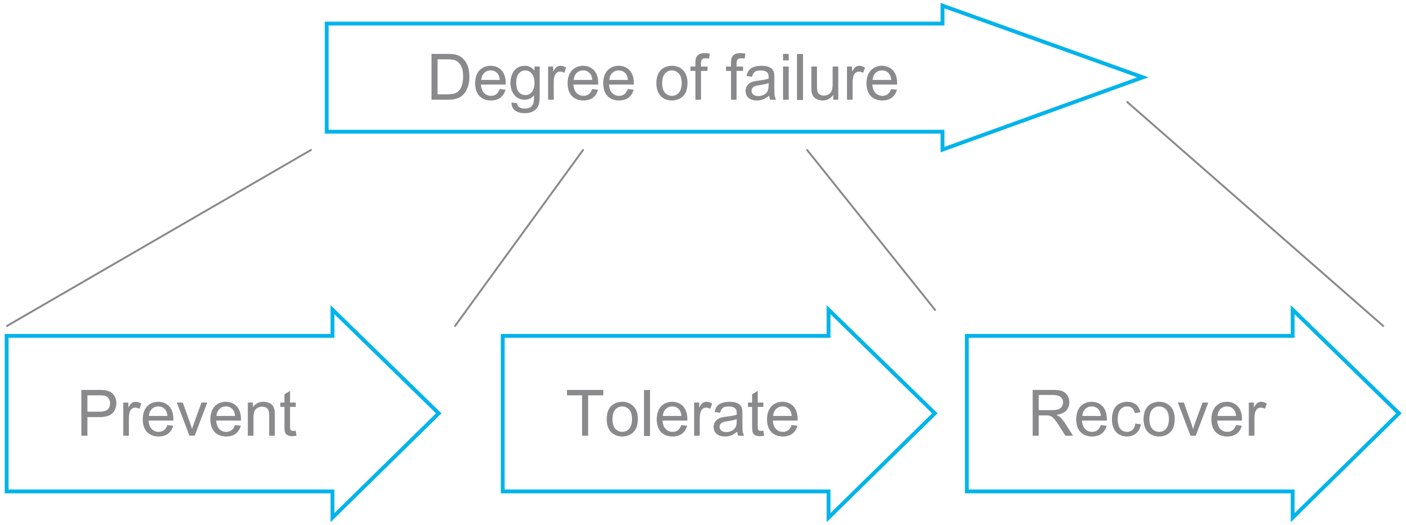


ISO 9126, Part 1
Reliability

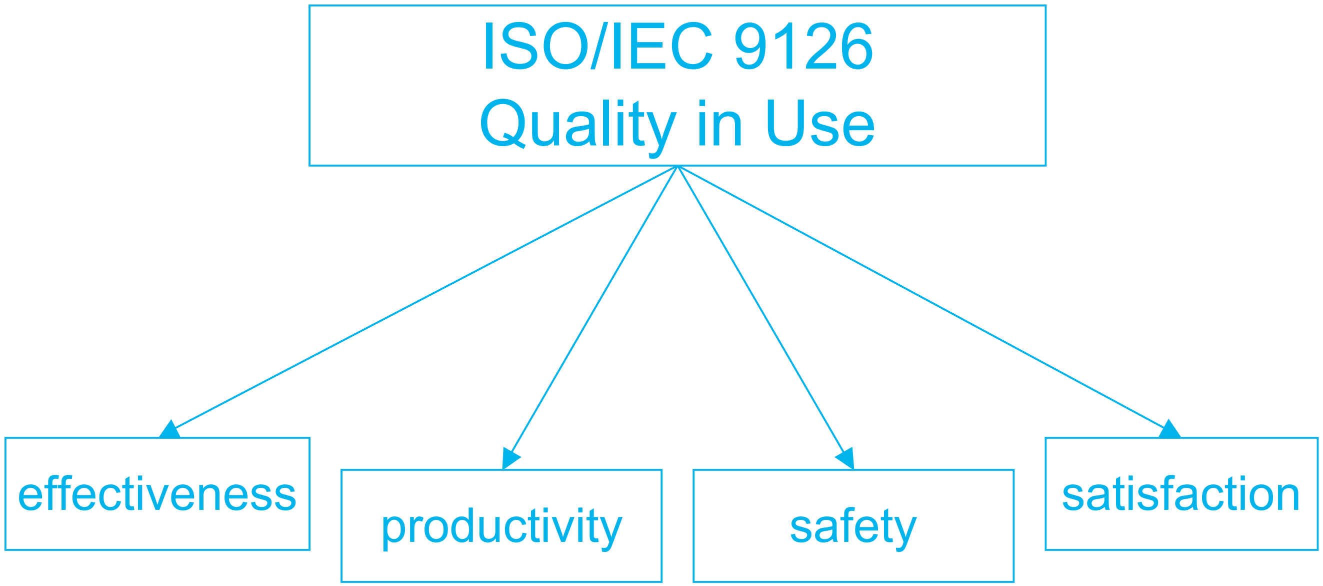
Software Improvement Group

Reliability =

- *Maturity*: how much has been done to prevent failures?
- *Fault tolerance*: when failure occurs, is it fatal?
- *Recoverability*: when fatal failure occurs, how much effort to restart?



ISO/IEC 9126, Part 1
Product quality model: quality-in-use



ISO 9126
Part 2,3: metrics

Software Improvement Group

External metrics, e.g.:

- Changeability: "change implementation elapse time", time between diagnosis and correction
- Testability: "re-test efficiency", time between correction and conclusion of test

Internal metrics, e.g.:

- Analysability: "activity recording", ratio between actual and required number of logged data items
- Changeability: "change impact", number of modifications and problems introduced by them

Critique

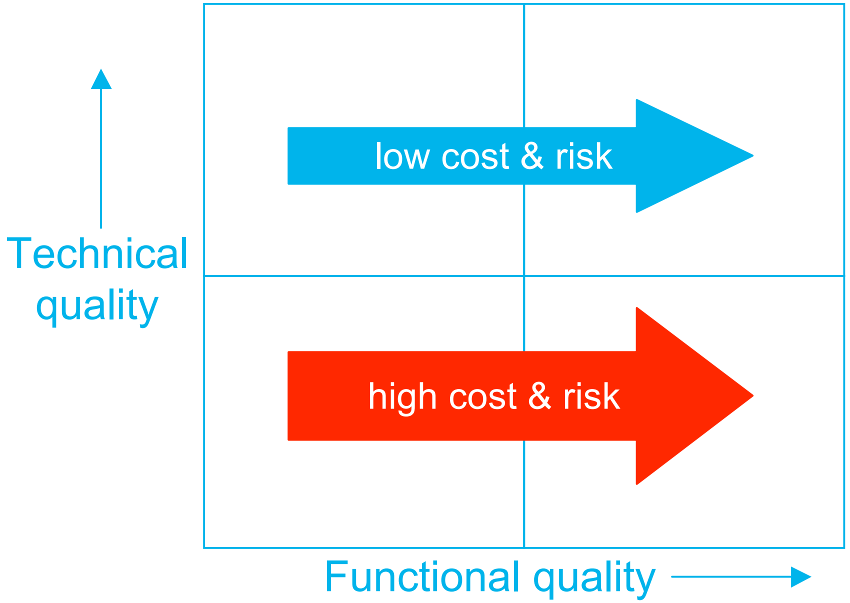
- Not pure *product* measures, rather *product in its environment*
- Measure *after* the fact
- No clear distinction between functional and technical quality

The issue

- Companies innovate and change
- Software systems need to adapt in the same pace as the business changes
- Software systems that do not adapt lose their value
- The technical quality of software systems is a key element

Functional *vs* technical quality

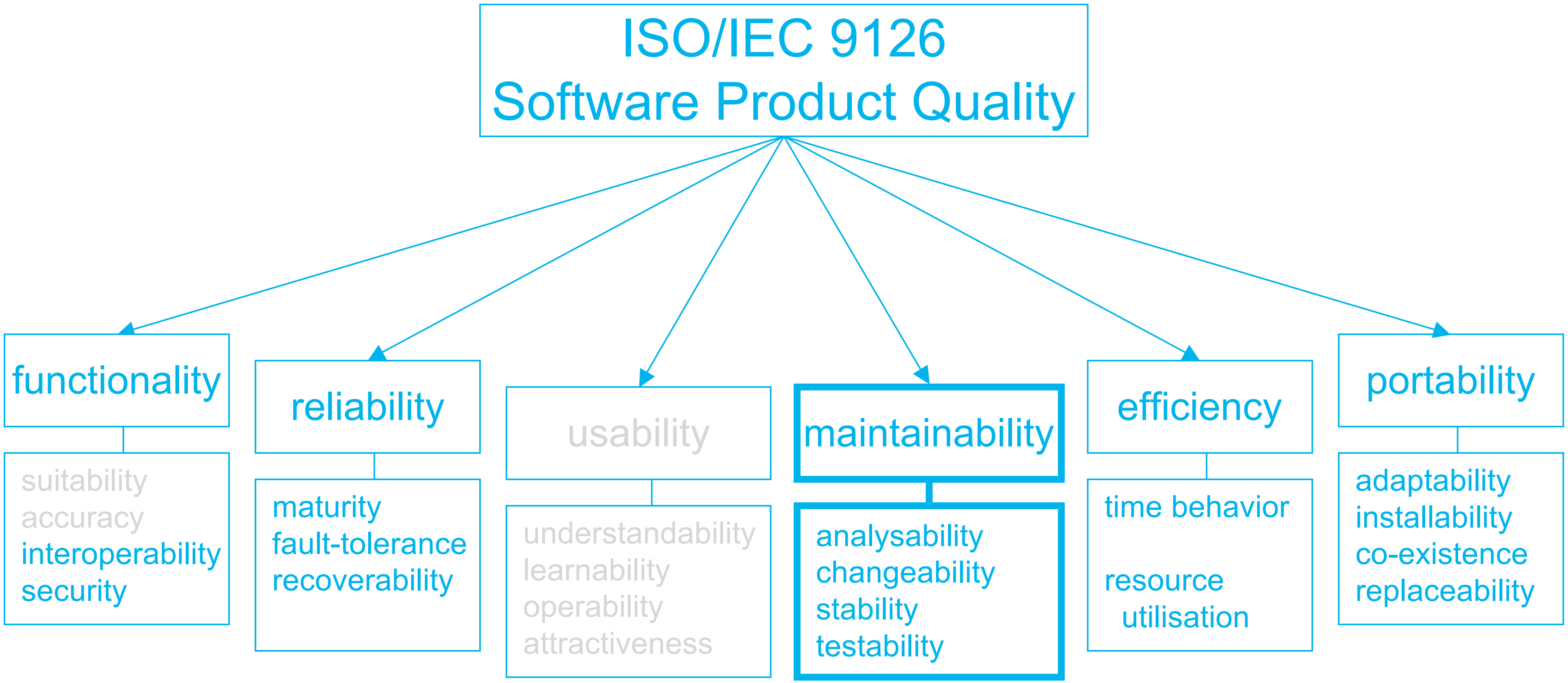
Software Improvement Group



Software with high technical quality can evolve with low cost and risk to keep meeting functional and non-functional requirements.

ISO/IEC 9126, Part 1
Product quality model: technical quality

Software Improvement Group



What is software quality? ✓

What are the functional and technical aspects of quality? ✓

How can technical quality be measured? ?

A Challenge

Software Improvement Group

Use source code metrics to measure technical quality?

Plenty of metrics defined in literature

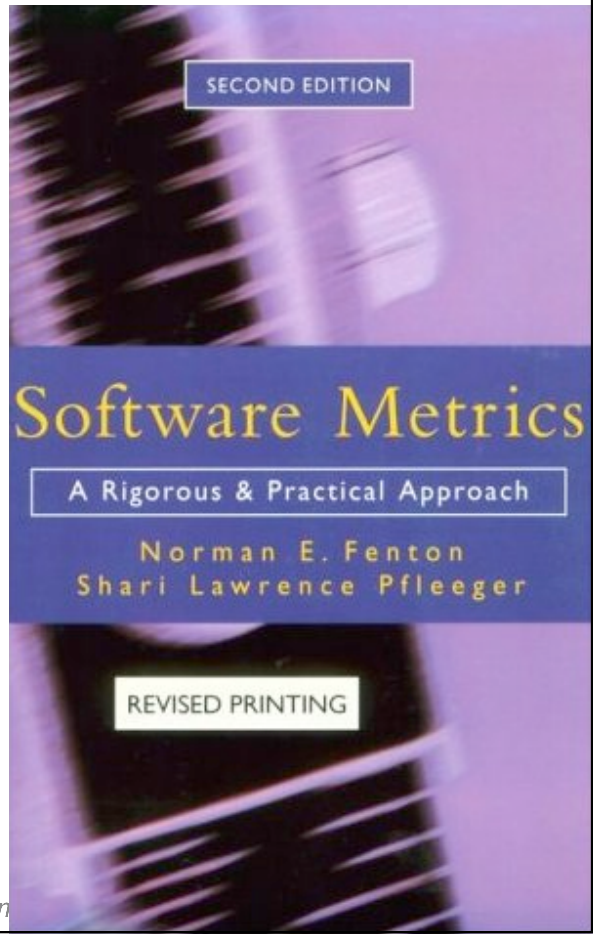
- LOC, cyclomatic complexity, fan in/out, coupling, cohesion, …
- Halstead, Chidamber-Kemener, Shepperd, …

Plenty of tools available

- Variations on Lint, PMD, FindBugs, …
- Coverity, FxCop, Fortify, QA-C, Understand, …
- Integrated into IDEs

But:

- Do they measure technical quality of a system?



Source code metrics
Lines of code (LOC)

Software Improvement Group

- Easy! Or …

- SLOC = Source Lines of Code
  - Physical (≈ newlines)
  - Logical (≈ statements)
- Blank lines, comment lines, lines with only “}”
- Generated *versus* manually written

- Measure effort / productivity: specific to programming language

Source code metrics
Function Point Analysis (FPA)

Software Improvement Group

- A.J. Albrecht - IBM - 1979
- Objective measure of functional size
- Counted manually
  - IFPUG, Nesma, Cocomo
  - Large error margins
- Backfiring
  - Per language correlated with LOC
  - SPR, QSM
- Problematic, but popular for estimation

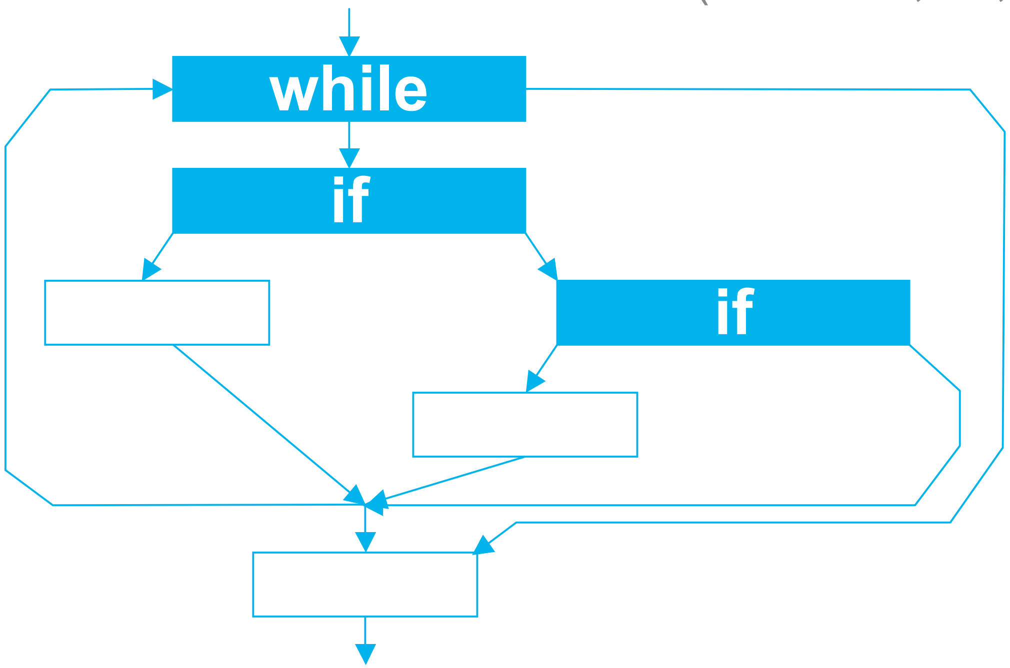
Table 2. Sample Function Point Calculations

Raw Data	Weights		Function Points
1 Input	X 4	=	4
1 Output	X 5	=	5
1 Inquiry	X 4	=	4
1 Data File	X 10	=	10
1 Interface	X 7	=	7

Unadjusted Total			30
Compexity Adjustment			None
Adjusted Function Points			30

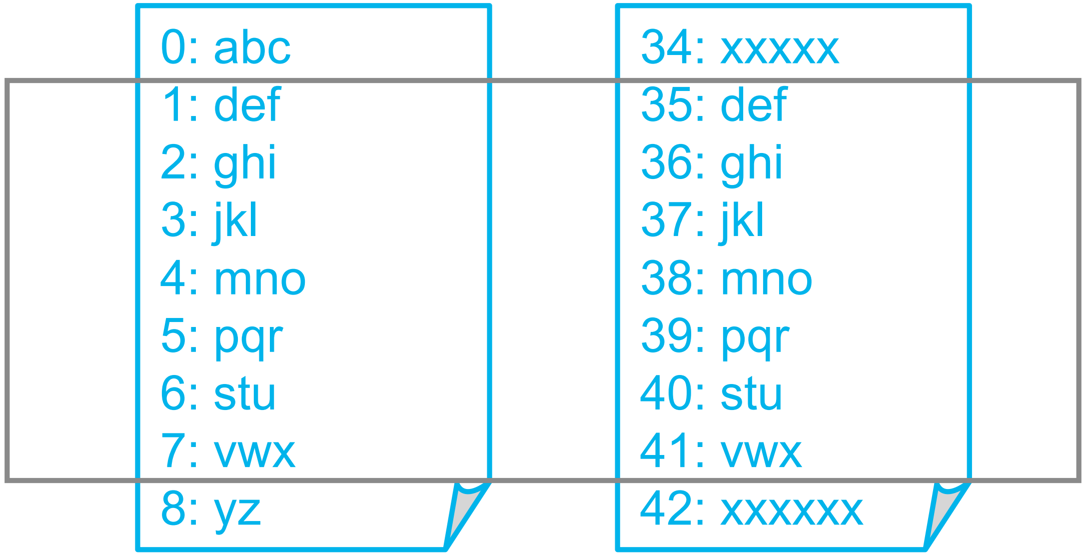
Source code metrics
Cyclomatic complexity

- T. McCabe, *IEEE Trans. on Sw Engineering*, 1976
- Accepted in the software community
- Number of independent, non-circular paths per method
- Intuitive: number of decisions made in a method
- 1 + the number of if statements (and while, for, ...)



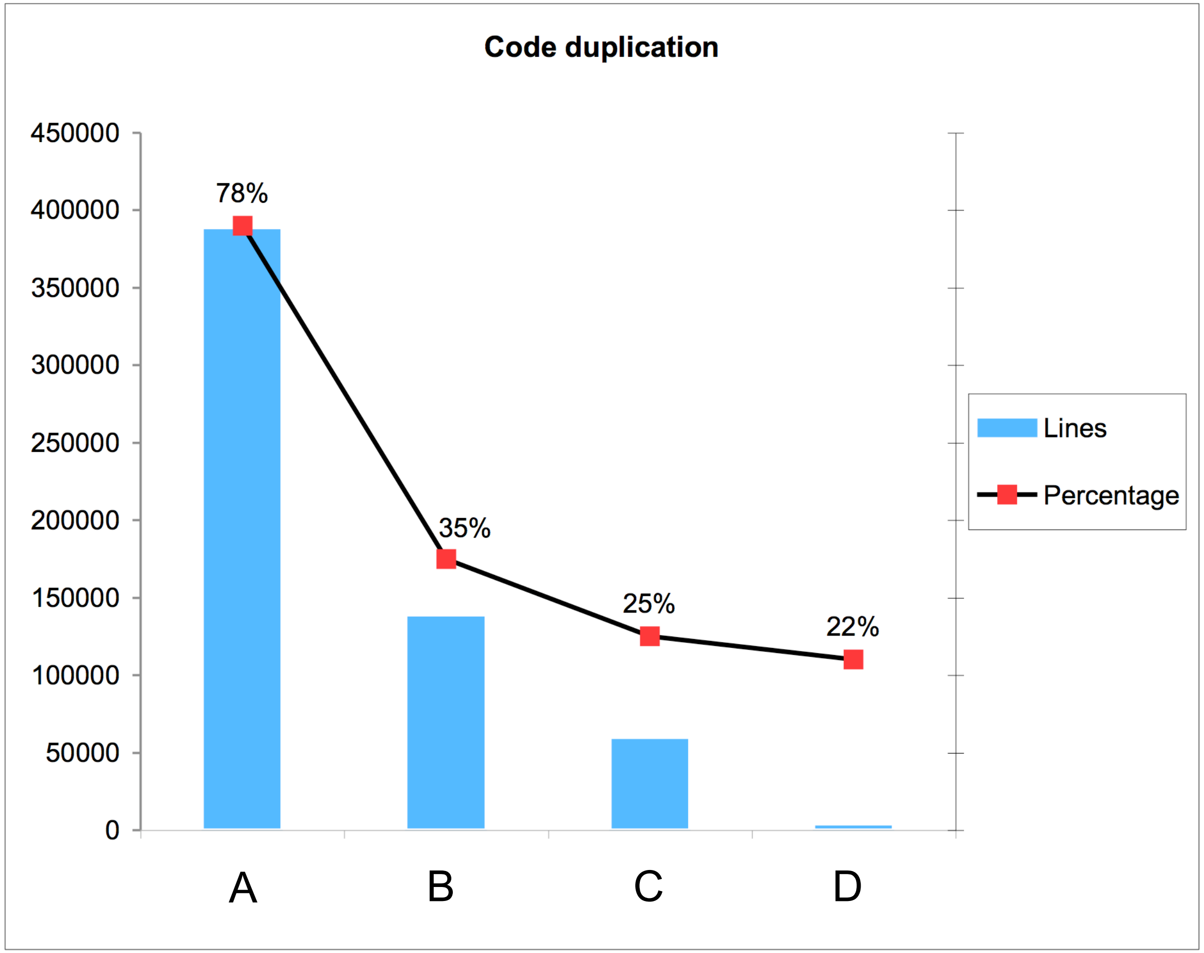
Software Improvement Group

Code duplication measurement



Number of duplicated lines: 14

Code duplication



Source code metrics
Coupling

Software Improvement Group

- Efferent Coupling (Ce)
  - How many classes do I depend on?
- Afferent Coupling (Ca)
  - How many classes depend on me?
- Instability = Ce/(Ca+Ce) ∈ [0,1]
  - Ratio of efferent *versus* total coupling
  - 0 = very stable = hard to change
  - 1 = very instable = easy to change

Figure 1. Coupling graph

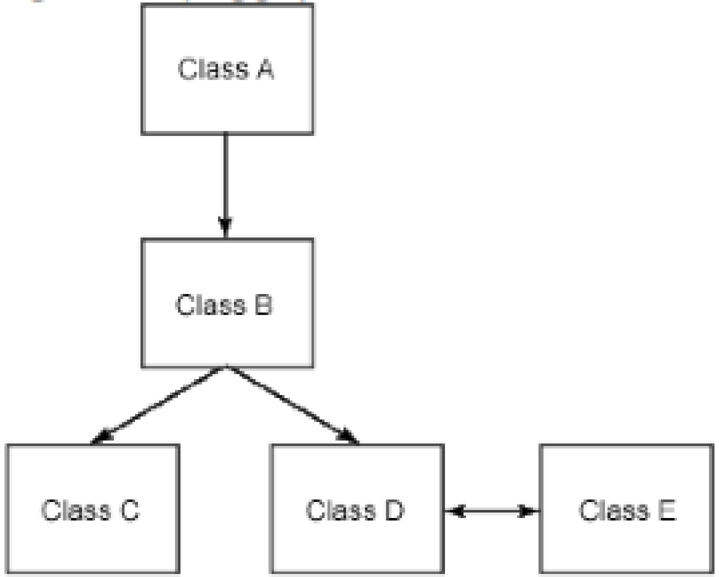


Table 1. Results of compiling a single class

Class to Compile	Other Classes Compiled	Afferent Couplings	Efferent Couplings	Instability Factor
A	B,C,D,E	0	4	1
B	C,D,E	1	3	0.75
C	-	2	0	0
D	E	3	1	0.25
E	D	3	1	0.25

Software metrics crisis

How does measurement data lead to information?

Software Improvement Group

Plethora of software metrics

- Ample definitions in literature
- Ample tools that calculate

Measurement yields data, not information

- How to aggregate individual measurement values?
- How to map aggregated values onto quality attributes?
- How to set thresholds?
- How to act on results?

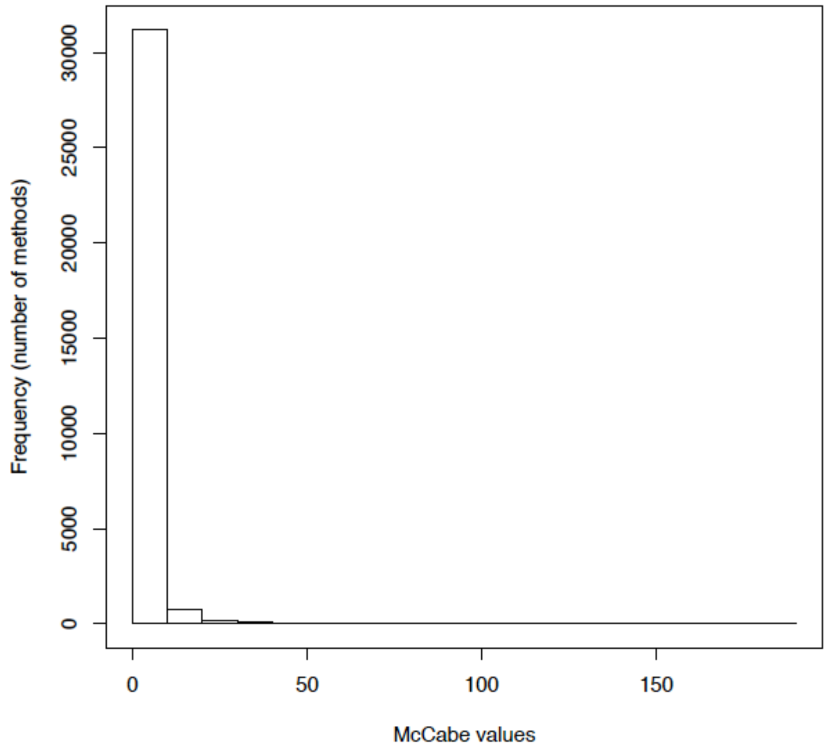
SIG quality model handles these issues in a pragmatic way

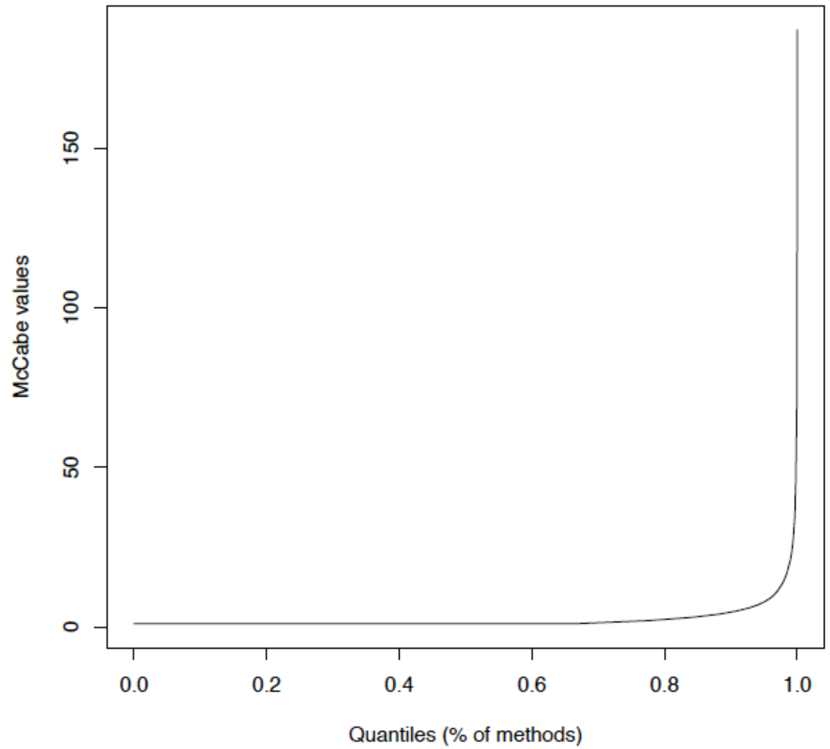
The statistical nature of software metrics
Averaging is fundamentally flawed

Software Improvement Group

Average

- Is measure for *central tendency*
- For “symmetric” distributions, such as *normal*. But:



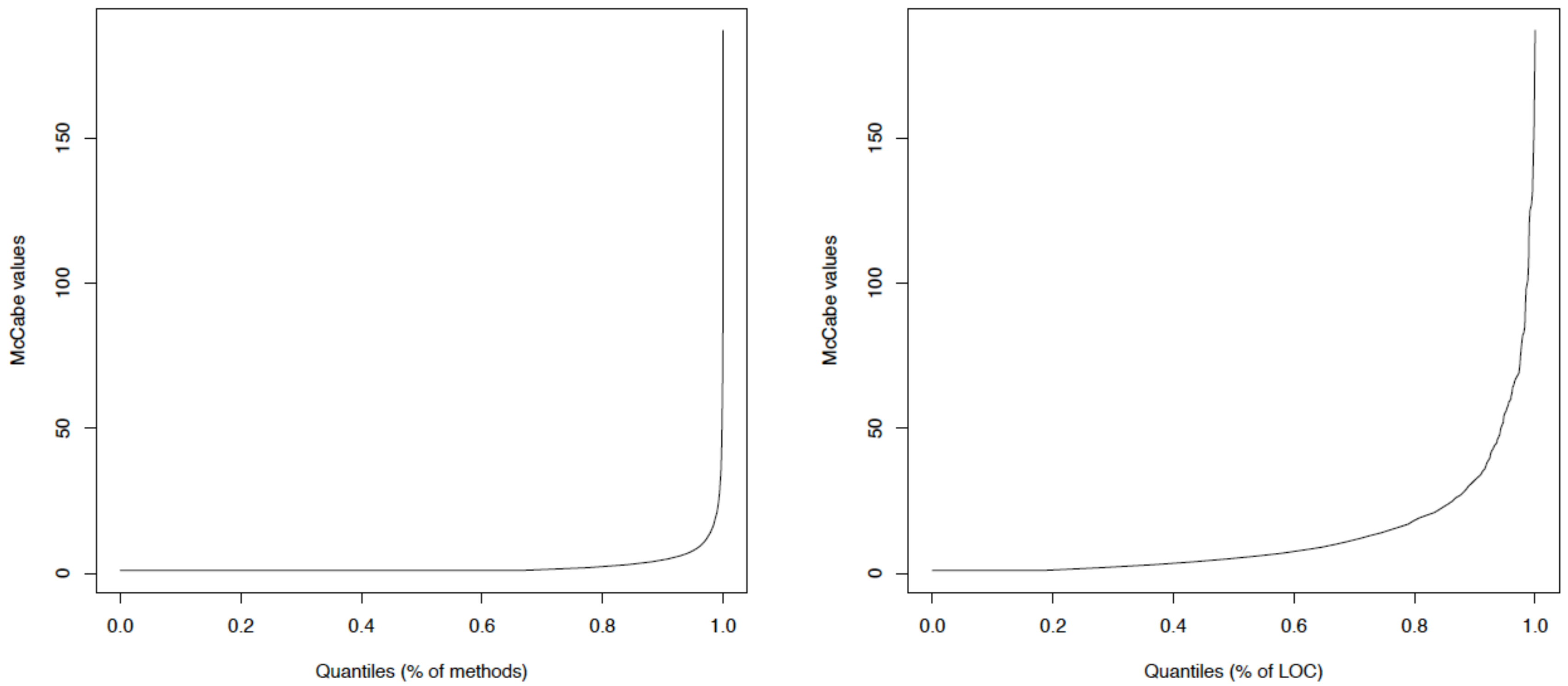


The statistical nature of software metrics
Emphasize area of risk

Software Improvement Group

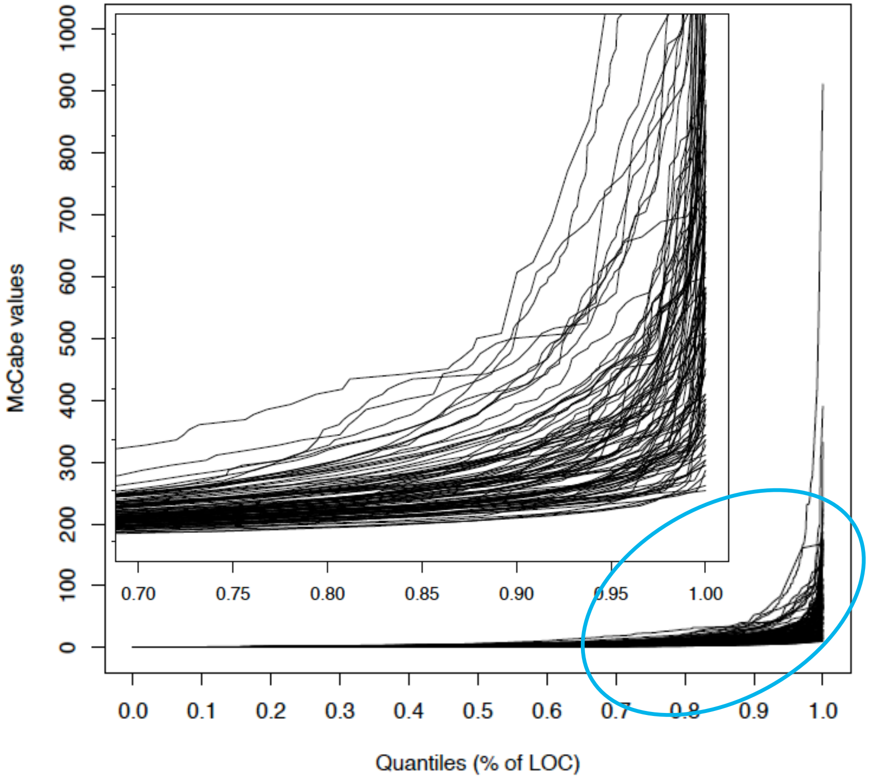
Exploit a-symmetry

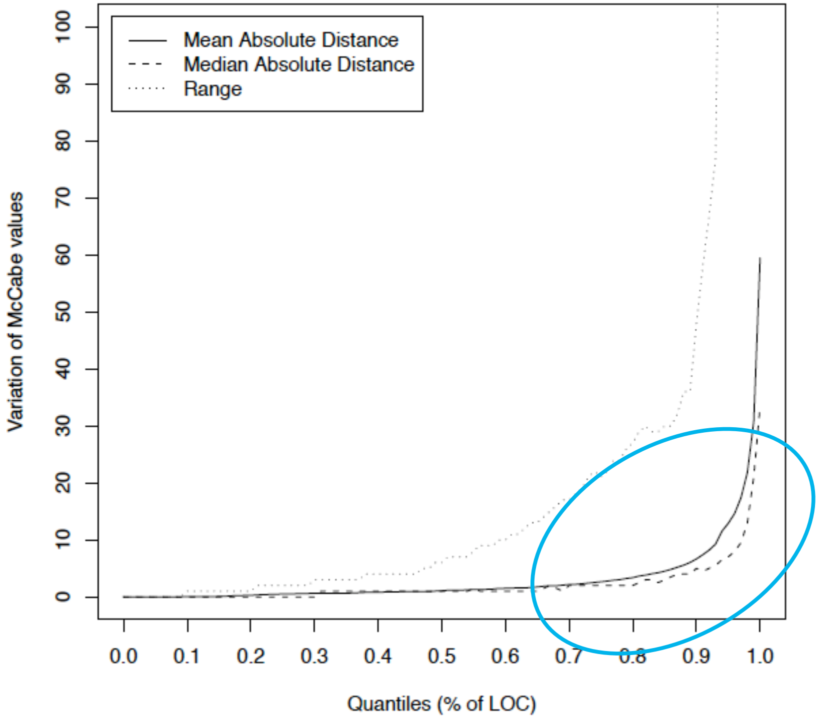
- High-risk code is on the right
- Weighing with LOC



The statistical nature of software metrics
Go where the variation is

Software Improvement Group



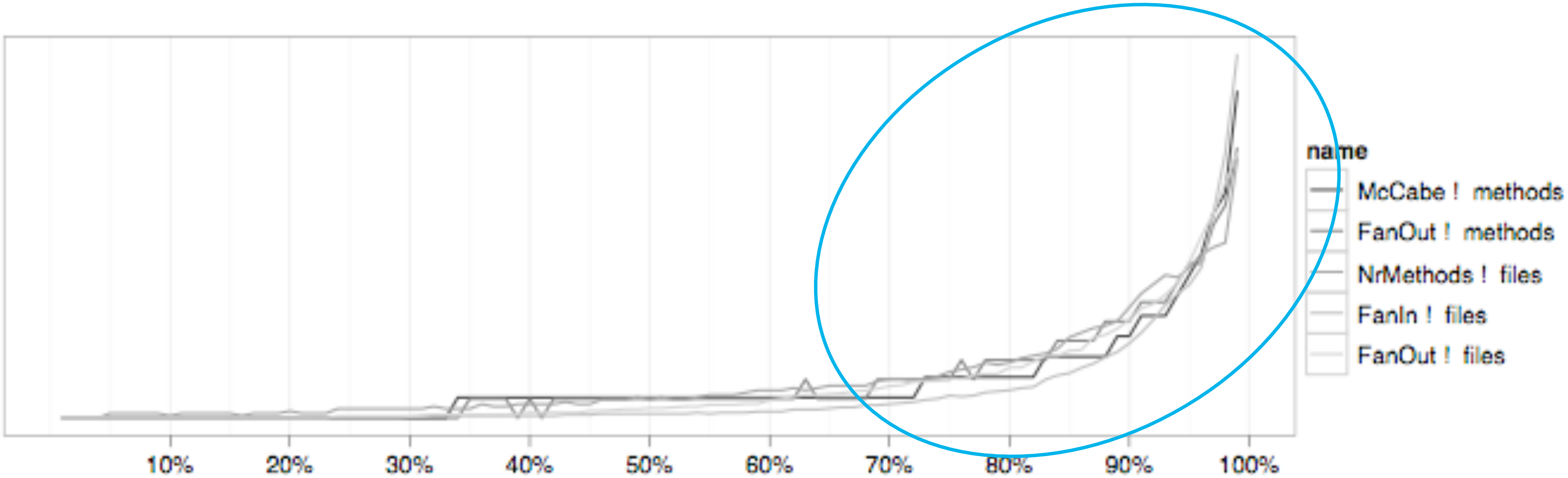


Observe for all:

- Systems are similar in low percentiles. Systems differ in higher percentiles.
- Interesting differences occur mostly above the 70% percentile

The statistical nature of software metrics
Go where the variation is

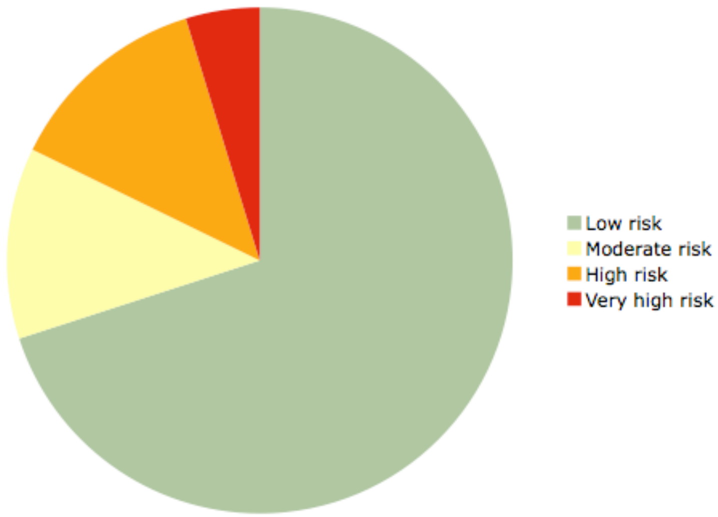
Similar for most source code metrics



SIG Quality Model
Quality profiles

Software Improvement Group

1. Measure source code metrics per method / file / module
2. Summarize distribution of measurement values in “Quality Profiles”

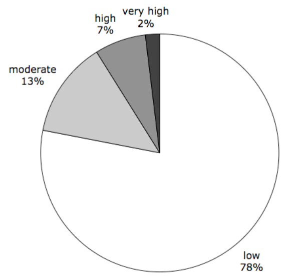


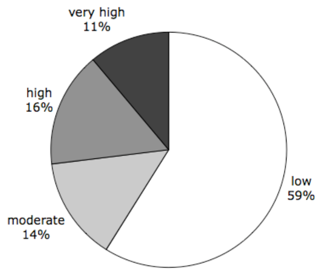
Cyclomatic complexity	Risk category
1 - 10	Low
11 - 20	Moderate
21 - 50	High
> 50	Very high

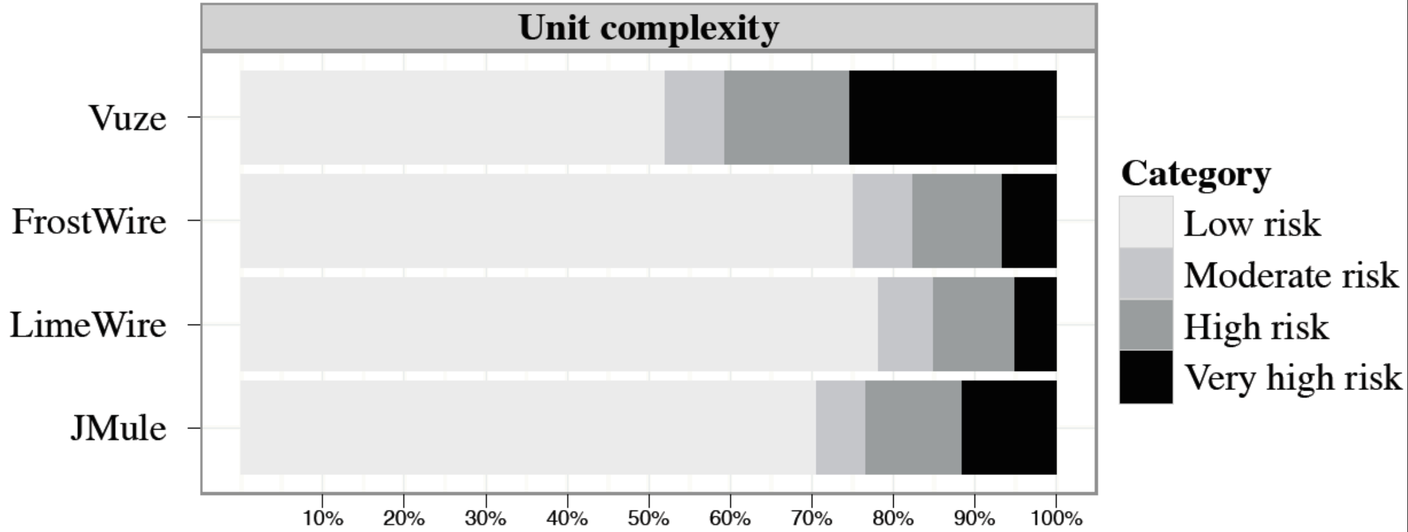
Sum lines of code per category

Lines of code per risk category			
Low	Moderate	High	Very high
70 %	12 %	13 %	5 %

Aggregation by averaging is fundamentally flawed







Quality profiles, in general

Software Improvement Group

Input

- type Input metric = Map item (metric,LOC)

Risk groups

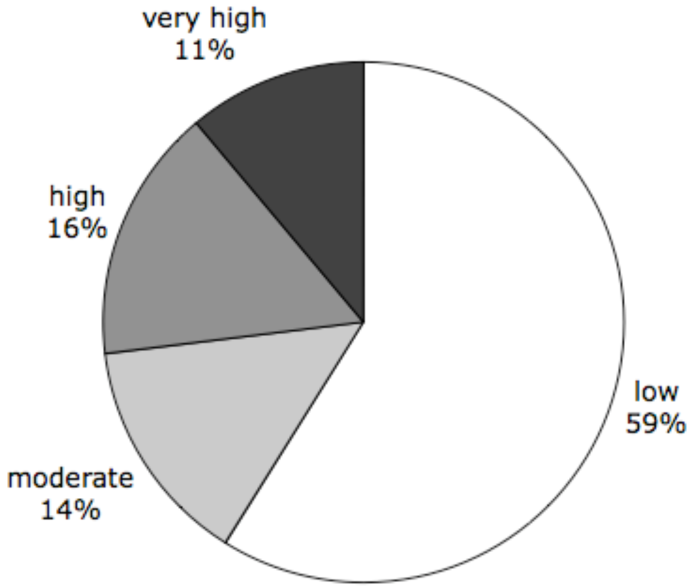
- type Risk = Low | Moderate | High | Very High
- risk :: metric → Risk

Output

- type ProfileAbs = Map Risk LOC
- type Profile = Map Risk Percentage

Aggregation

- profile :: Input metric → Profile



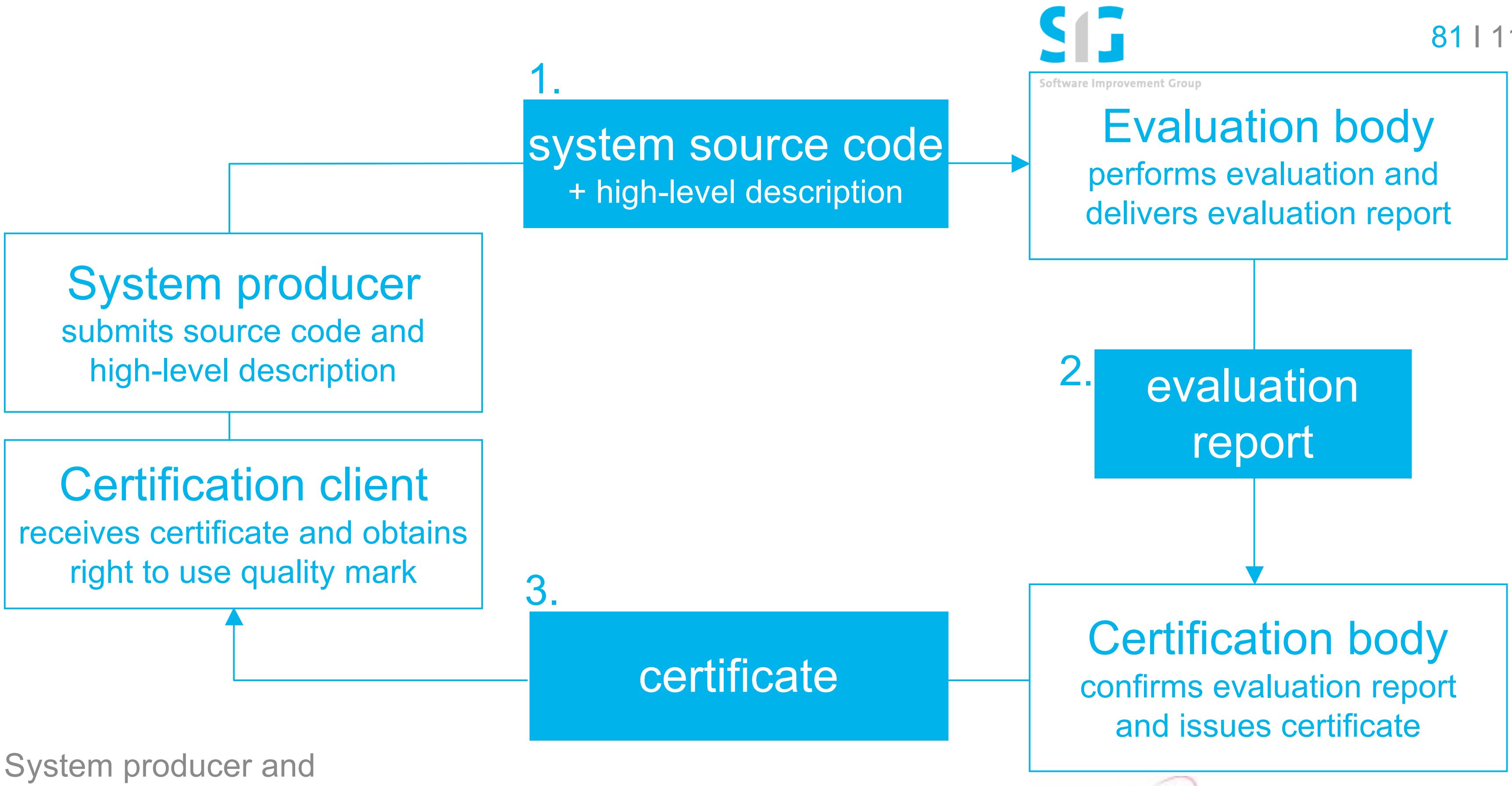
SIG Quality Model
How do measurements lead to ratings?

Software Improvement Group

A practical model for measuring maintainability
Heitlager, Kuipers, Visser in QUATIC 2007, IEEE Press

a. Aggregate measurements into “Quality Profiles”
b. Map measurements and quality profiles to ratings for system properties
c. Map ratings for system properties to ratings for ISO/IEC 9126 quality characteristics
d. Map to overall rating of technical quality

Software product certification
by SIG and TÜViT



System producer and certification client can be the same organization

Evaluation results

Software Improvement Group

Evaluation report

- Defines scope of the evaluation
- Summarizes measurement results
- Provides ratings (properties, quality, and overall)
- May provide hints for the producer to improve ratings

Certificate

- States conformance to *SIG/TÜViT Evaluation Criteria*
- Confers right to use quality mark "TÜViT Trusted Product *Maintainability*"

Main-tainability
TÜViT®
2009 Trusted Product
Voluntary Validation

Further reading

Software Improvement Group

A pragmatic model for measuring maintainability.
Heitlager, T. Kuipers, J. Visser. QUATIC 2007.

Certification of Technical Quality of Software.
J.P. Correia, J.Visser. OpenCert 2008.

Mapping System Properties to ISO/IEC 9126 Maintainability Characteristics
J.P. Correia, Y. Kanellopoulos, J.Visser. SQM 2009.

Software Risk Assessment service

Software Improvement Group

Assignment

- “Can we scale from 100 to 100,000 customers?”
- “Should we accept delay and cost overrun, or cancel the project?”

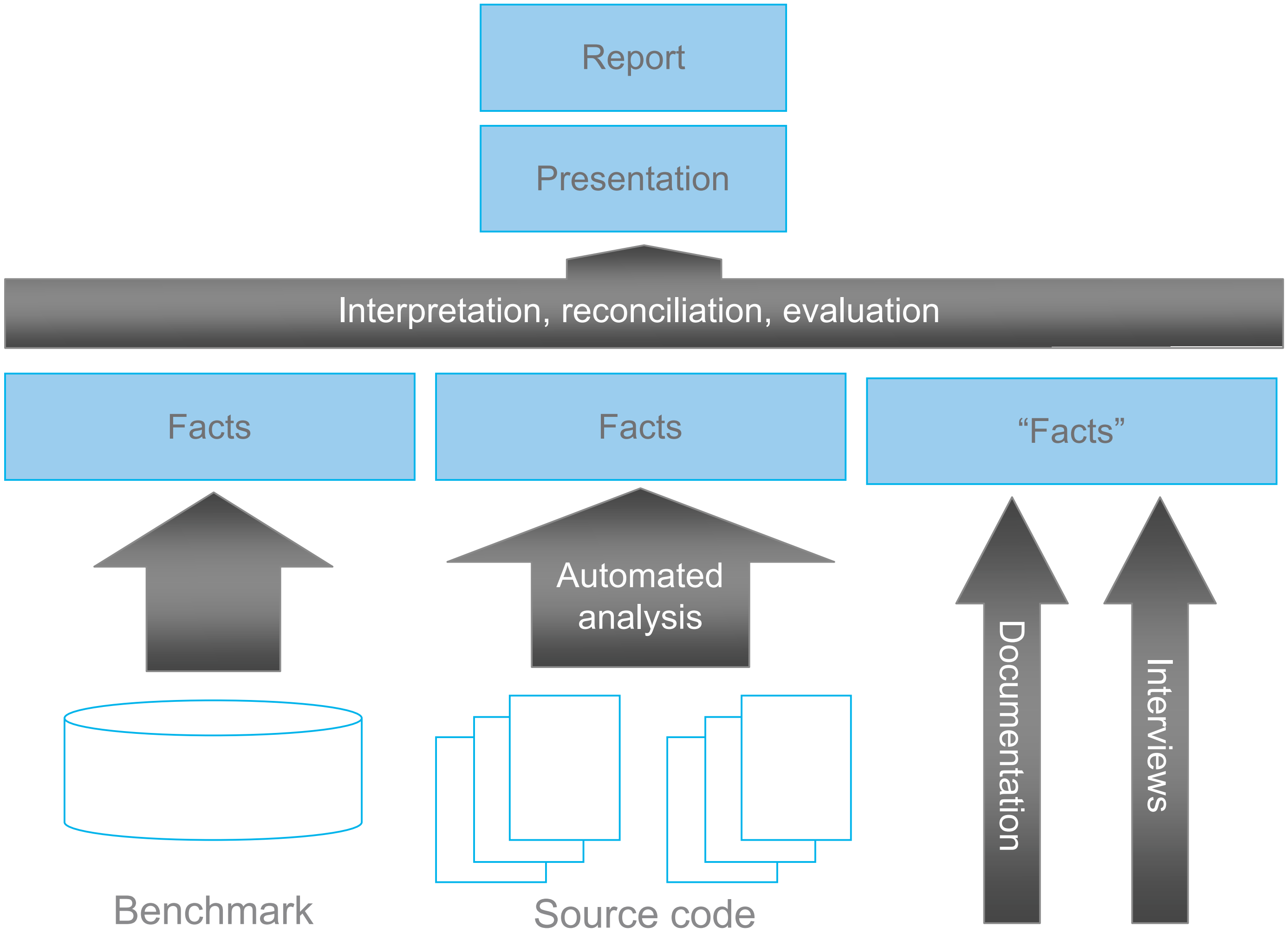
Analysis

- Source code: understanding (reverse engineering) + evaluation (quality)
- Interviews: technical + strategic

Reporting

- Quality judgment using star ratings
- Risk analysis putting quality findings in business perspective
- Recommendations to mitigate risks

Software Risk Assessment



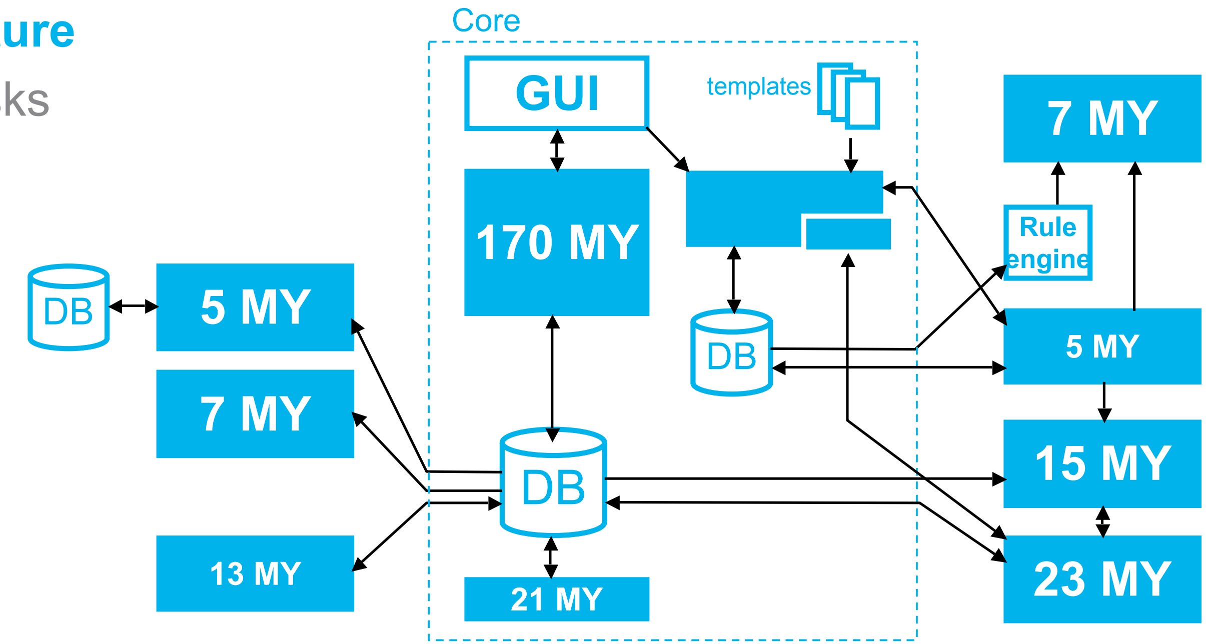
Software Risk Assessment

Example: stagnation before go-live

Software Improvement Group

Internal architecture

- Technology risks
- Rebuild value
- Quality



Results

- Insurmountable stability issues, untestable, excessive maintenance burden
- Now: reduce technical complexity, partially automate deployment
- Start planning replacement

Software Monitoring service

Software Improvement Group

Quality roadmap

- "complexity from 2 to 4 stars by 3rd month" in maintenance project
- "final product shall be 4 stars" in development project

Dashboard

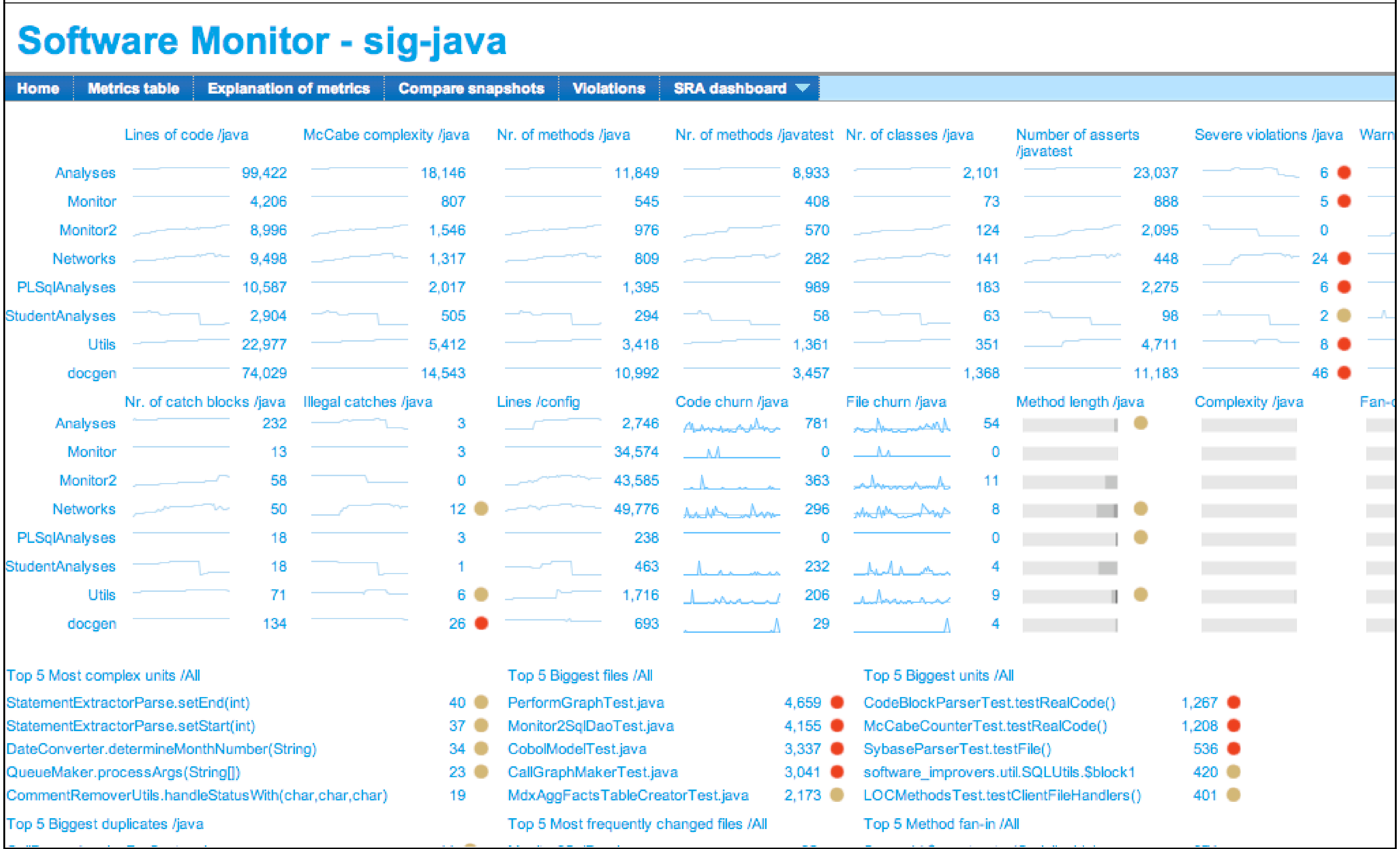
- Regular analysis of source code typically once per week
- Shown on dashboard with overviews and drill down possibilities

Consultancy

- Regular reports (presentation and/or written)
- Guard quality agreements, meet quality targets.
- Identify risks and opportunities

Software Monitor
Dashboard

Software Improvement Group



Software Monitor - sig-java

Home | Metrics table | Explanation of metrics | Compare snapshots | Violations | SRA dashboard

	Lines of code /java	McCabe complexity /java	Nr. of methods /java	Nr. of methods /javatest	Nr. of classes /java	Number of asserts /javatest	Severe violations /java	Warn
Analyses	99,422	18,146	11,849	8,933	2,101	23,037	6	
Monitor	4,206	807	545	408	73	888	5	
Monitor2	8,996	1,546	976	570	124	2,095	0	
Networks	9,498	1,317	809	282	141	448	24	
PLSqlAnalyses	10,587	2,017	1,395	989	183	2,275	6	
StudentAnalyses	2,904	505	294	58	63	98	2	
Utils	22,977	5,412	3,418	1,361	351	4,711	8	
docgen	74,029	14,543	10,992	3,457	1,368	11,183	46	

	Nr. of catch blocks /java	Illegal catches /java	Lines /config	Code churn /java	File churn /java	Method length /java	Complexity /java	Fan-o
Analyses	232	3	2,746	781	54			
Monitor	13	3	34,574	0	0			
Monitor2	58	0	43,585	363	11			
Networks	50	12	49,776	296	8			
PLSqlAnalyses	18	3	238	0	0			
StudentAnalyses	18	1	463	232	4			
Utils	71	6	1,716	206	9			
docgen	134	26	693	29	4			

Top 5 Most complex units /All	
StatementExtractorParse.setEnd(int)	40
StatementExtractorParse.setStart(int)	37
DateConverter.determineMonthNumber(String)	34
QueueMaker.processArgs(String[])	23
CommentRemoverUtils.handleStatusWith(char,char,char)	19

Top 5 Biggest files /All	
PerformGraphTest.java	4,659
Monitor2SqlDaoTest.java	4,155
CobolModelTest.java	3,337
CallGraphMakerTest.java	3,041
MdxAggFactsTableCreatorTest.java	2,173

Top 5 Biggest units /All	
CodeBlockParserTest.testRealCode()	1,267
McCabeCounterTest.testRealCode()	1,208
SybaseParserTest.testFile()	536
software_improvers.util.SQLUtils.$block1	420
LOCMethodsTest.testClientFileHandlers()	401

Top 5 Biggest duplicates /java

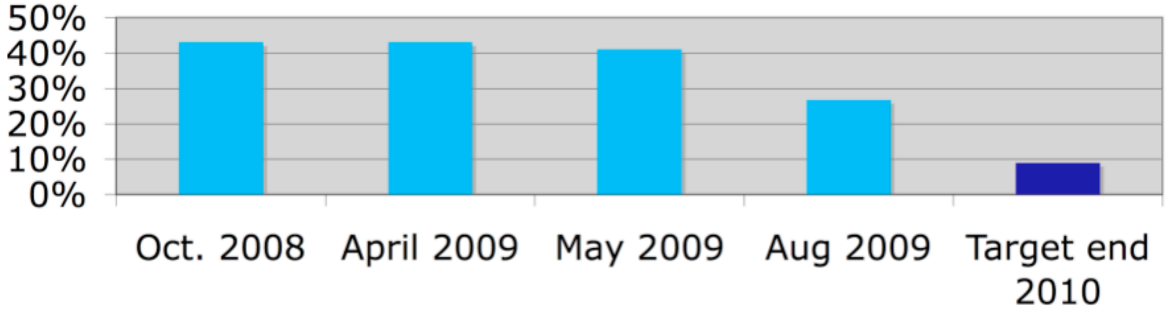
Top 5 Most frequently changed files /All

Top 5 Method fan-in /All

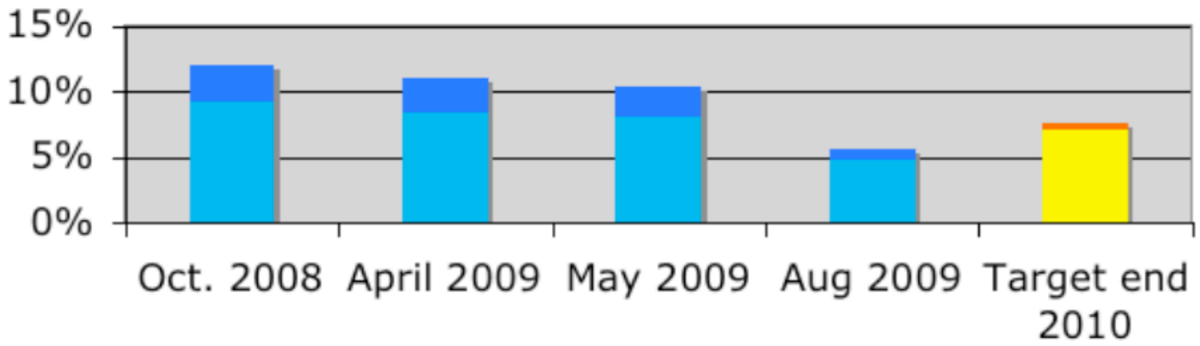
Software Monitor
Example: vendor management and roadmap

Software Improvement Group

Duplication



Complexity



From client testimonial:

- "Technical quality: as it improves adding functionality is made easier"
- "As quality was increasing, productivity was going up"

What should you remember (so far) from this lecture?

Testing

- Automated unit testing!

Patterns

- Run tools!

Quality and metrics

- Technical quality matters in the long run
- A few simple metrics are sufficient
- If aggregated in well-chosen, meaningful ways
- The simultaneous use of distinct metrics allows zooming in on root causes