Software Improvement Group

Measuring Software Product Quality

Eric Bouwers

June 20, 2013

T +31 20 314 0950
info@sig.eu
www.sig.eu

Software Improvement Group

Who are we?

- Highly specialized advisory company for cost, quality and risks of software
- Independent and therefore able to give objective advice

What do we do?

- Fact-based advice supported by our automated toolset for source code analysis
- Analysis across technologies by use of technology-independent methods

Our mission:

We give you control over your software.

Services

Software Risk Assessment

- In-depth investigation of software quality and risks
- Answers specific research questions

Software Monitoring

- Continuous measurement, feedback, and decision support
- Guard quality from start to finish

Software Product Certification

- Five levels of technical quality
- Evaluation by SIG, certification by TÜV Informationstechnik

Application Portfolio Analyses

- Inventory of structure and quality of application landscape
- Identification of opportunities for portfolio optimization

Today's topic

Measuring Software Product Quality

Some definitions

Projects

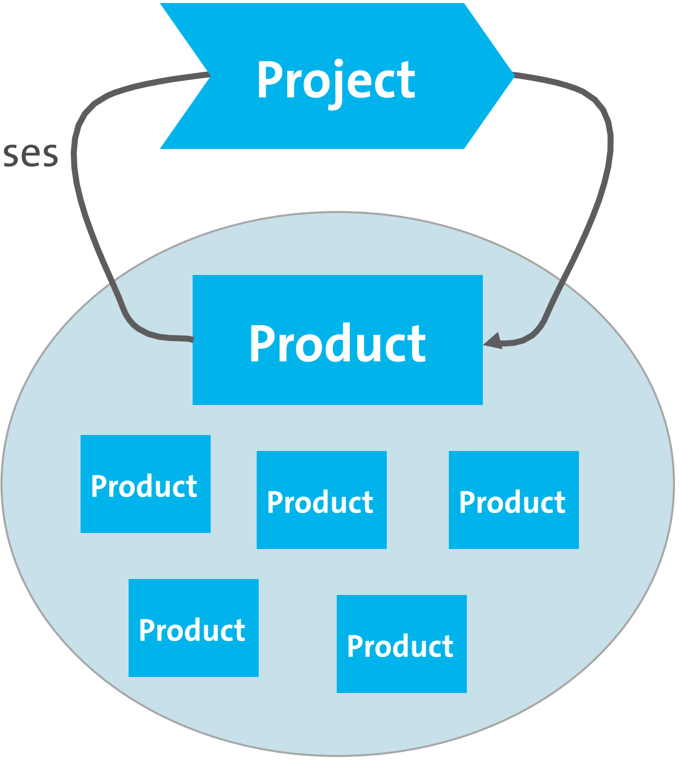
- Activity to create or modify software products
- Design, Build, Test, Deploy

Product

- Any software artifact produced to support business processes
- Either built from scratch, from reusable components, or by customization of “standard” packages

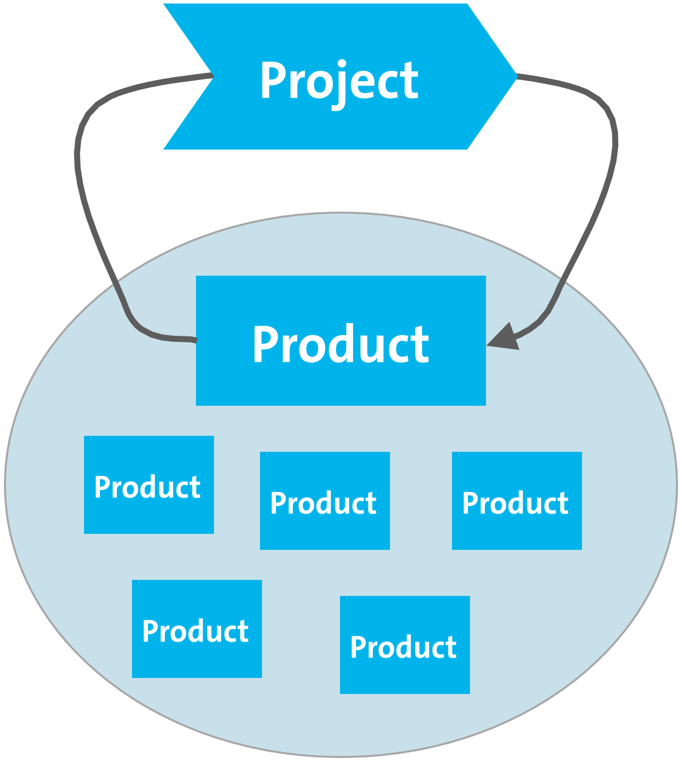
Portfolio

- Collection of software products in various phases of lifecycle
- Development, acceptance, operation, selected for decommissioning



Our claim today

“Measuring the quality of software products is key to successful software projects and healthy software portfolios”

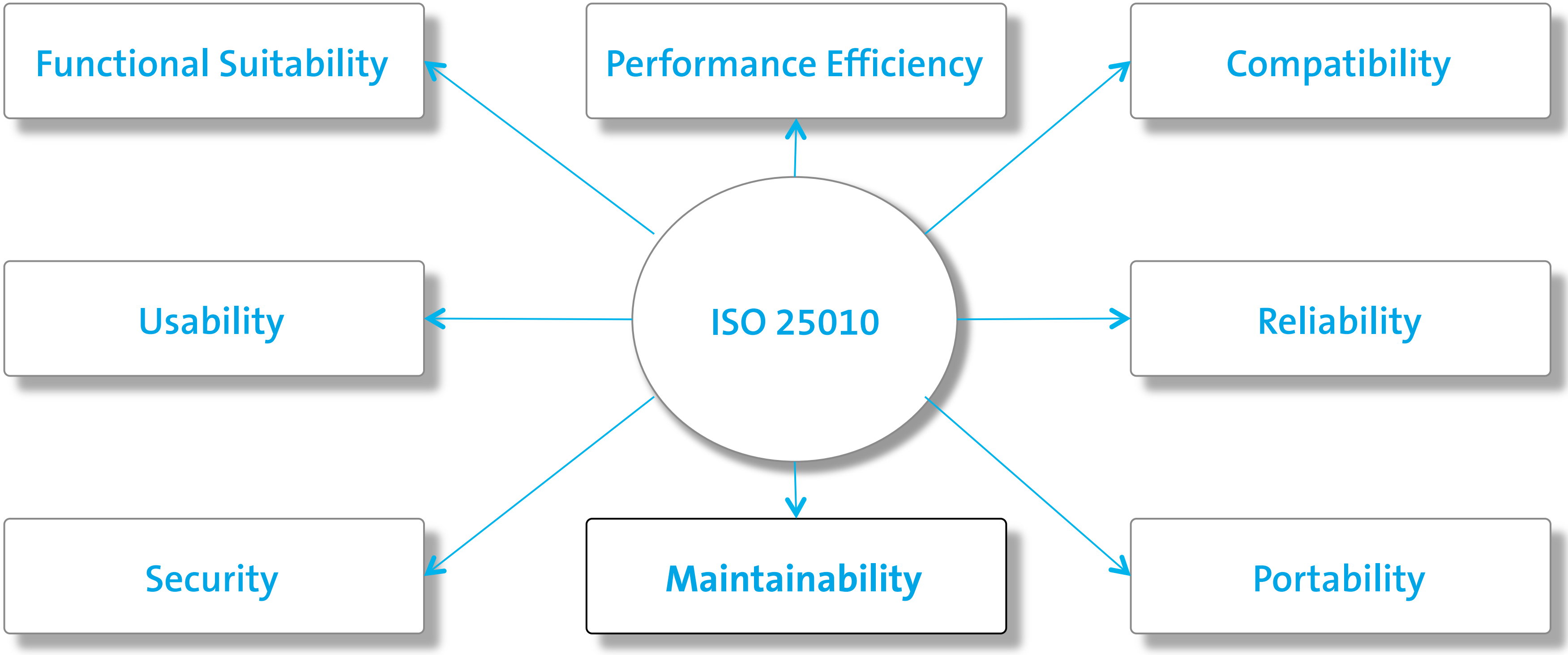


Today's topic

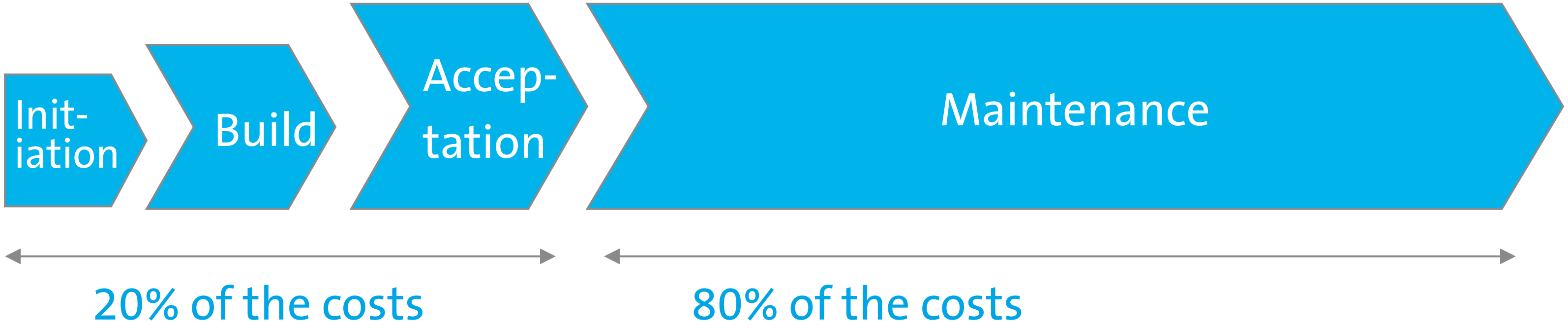
Measuring Software Product *Quality*

The ISO 25010 standard for software quality

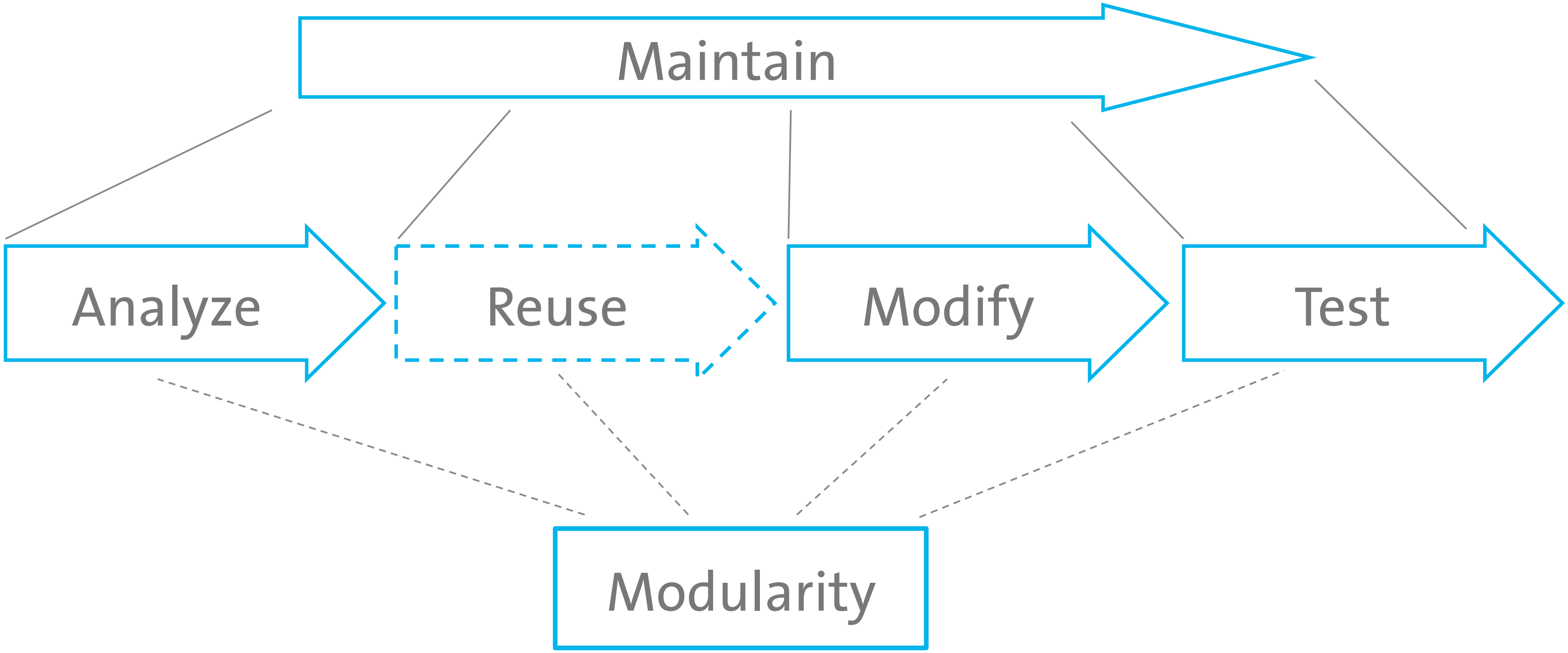
Software Improvement Group



Why focus on maintainability?



The sub-characteristics of maintainability in ISO 25010



Measuring maintainability
Some requirements

- Simple to understand
- Allow root-cause analyses
- Technology independent
- Easy to compute

Suggestions?

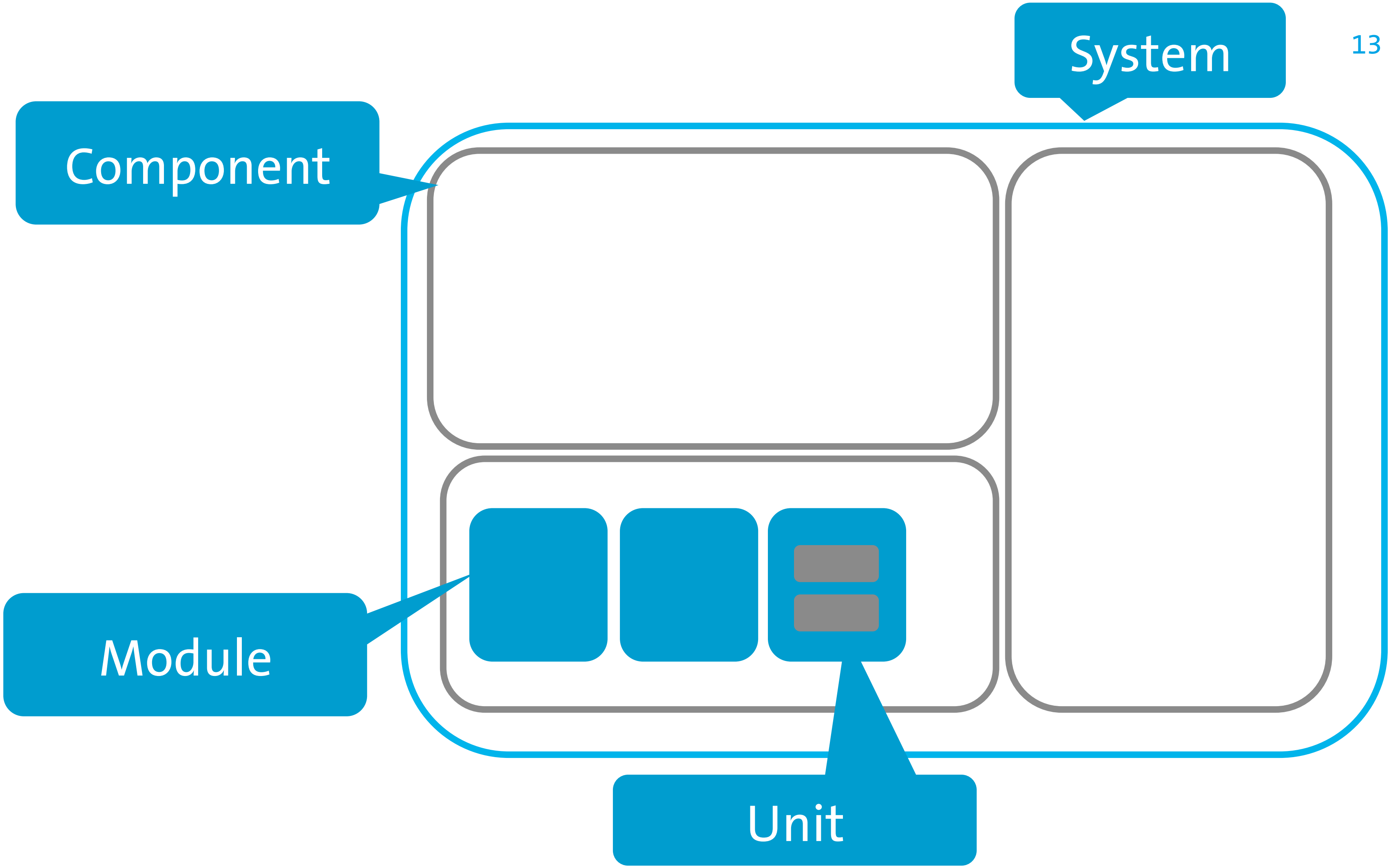
Heitlager, et. al. *A Practical Model for Measuring Maintainability, QUATIC 2007*

Measuring ISO 25010 maintainability using the SIG model

	Volume	Duplication	Unit size	Unit complexity	Unit interfacing	Module coupling	Component balance	Component independence
Analysability	X	X	X				X	
Modifiability		X		X		X		
Testability	X			X				X
Modularity						X	X	X
Reusability			X		X			

Measuring maintainability

Different levels of measurement



Source code measurement
Volume

Lines of code

- Not comparable between technologies

Function Point Analysis (FPA)

- A.J. Albrecht - IBM - 1979
- Counted manually
- Slow, costly, fairly accurate

Backfiring

- Capers Jones - 1995
- Convert LOC to FPs
- Based on statistics per technology
- Fast, but limited accuracy

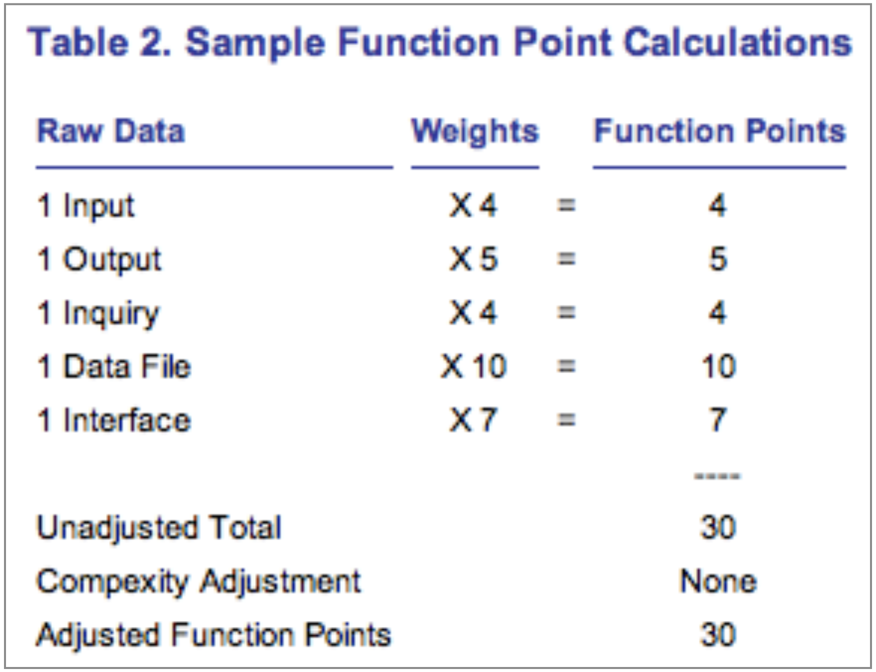


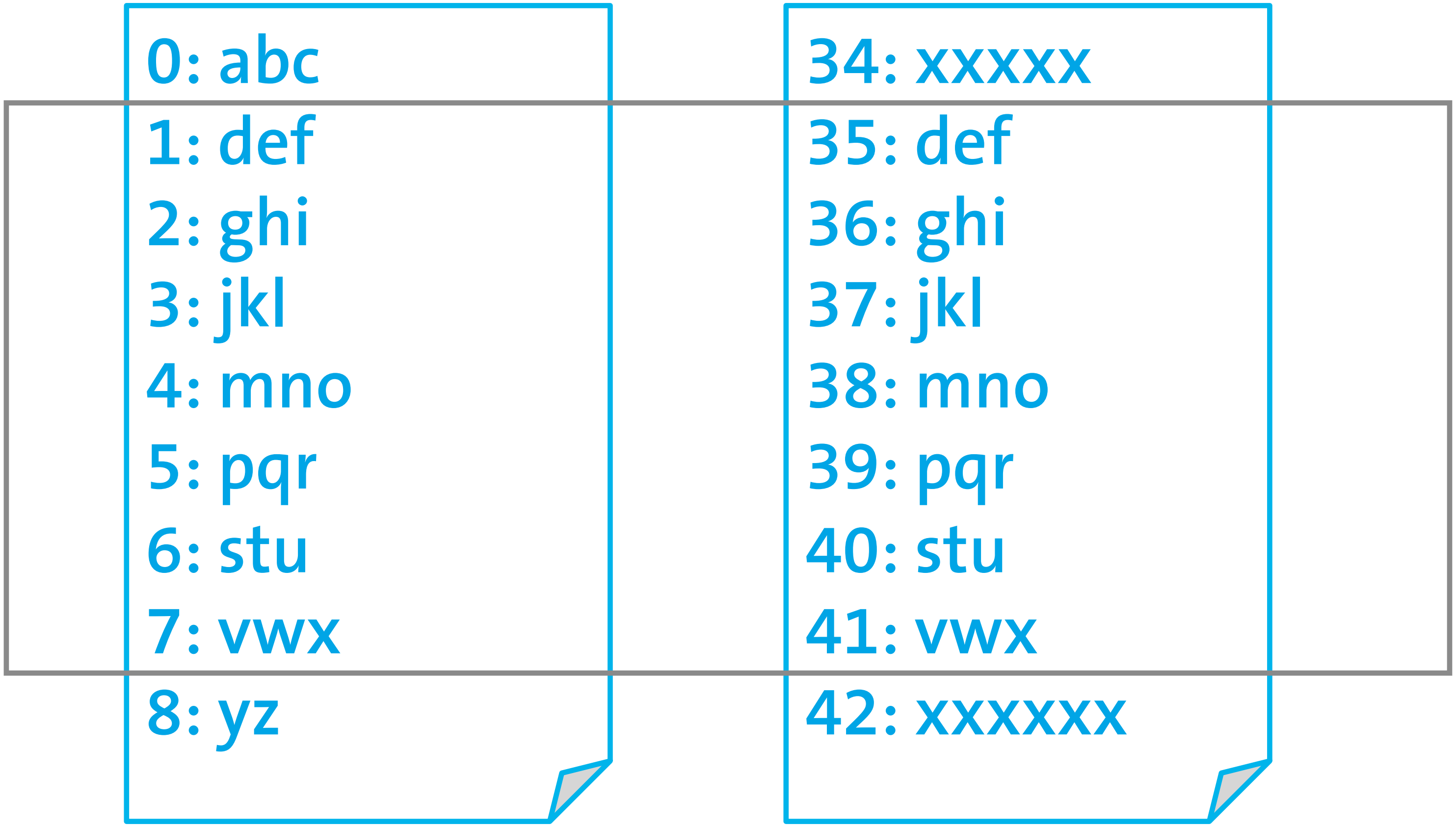
Table 2. Sample Function Point Calculations

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 measurement

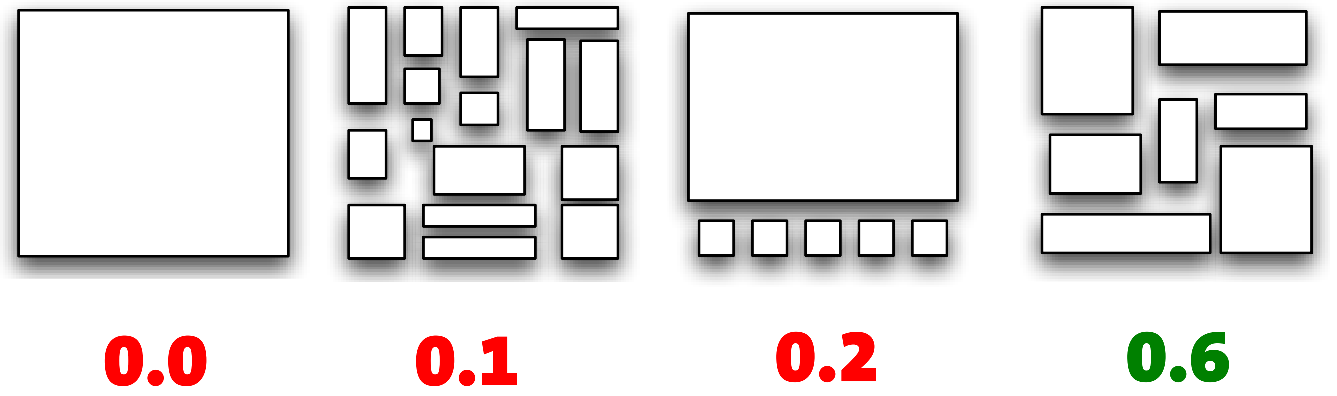
Duplication



0: abc
1: def
2: ghi
3: jkl
4: mno
5: pqr
6: stu
7: vwx
8: yz

34: xxxxx
35: def
36: ghi
37: jkl
38: mno
39: pqr
40: stu
41: vwx
42: xxxxxx

Source code measurement
Component balance

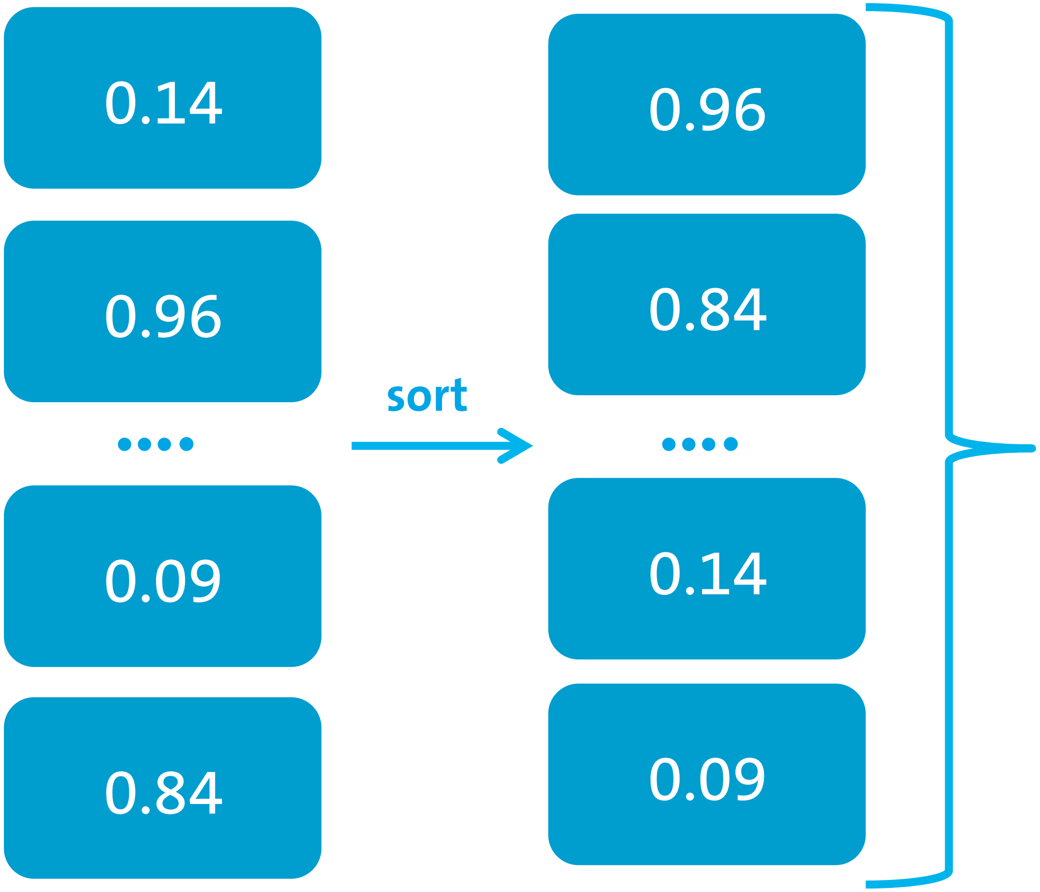


Measure for number and relative size of architectural elements

- CB = SBO × CSU
- SBO = system breakdown optimality, computed as distance from ideal
- CSU = component size uniformity, computed with Gini-coefficient

E. Bouwers, J.P. Correia, and A. van Deursen, and J. Visser, *A Metric for Assessing Component Balance of Software Architectures* in the proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA 2011)

From measurement to rating
A benchmark based approach



Threshold	Score
0.9	★★★★★
0.8	★★★★☆
0.5	★★★☆☆
0.3	★★☆☆☆
0.1	★☆☆☆☆

0.34

★★☆☆☆

Note: example thresholds

But what about the measurements on lower levels?

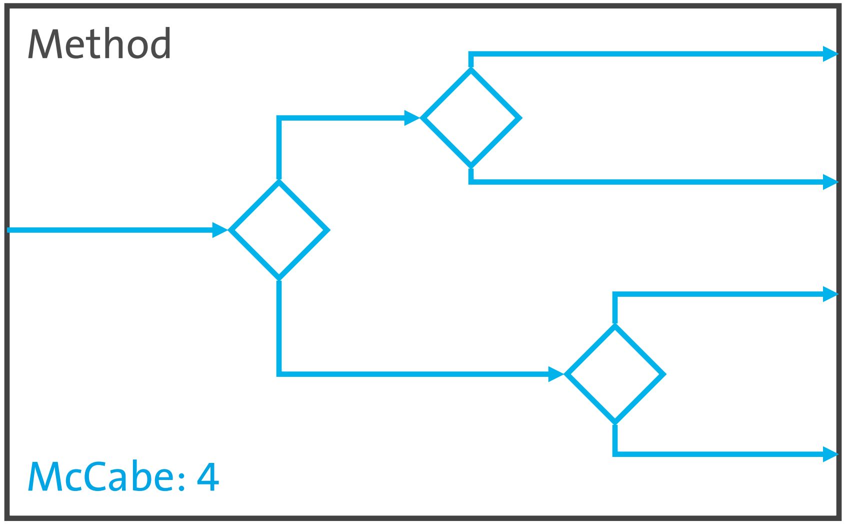
Software Improvement Group

	Volume	Duplication	Unit size	Unit complexity	Unit interfacing	Module coupling	Component balance	Component independence
Analysability	X	X	X				X	
Modifiability		X		X		X		
Testability	X			X				X
Modularity						X	X	X
Reusability			X		X			

Source code measurement
Logical complexity

- T. McCabe, *IEEE Transactions on Software Engineering*, 1976

- Academic: number of independent paths per method
- Intuitive: number of decisions made in a method
- Reality: the number of if statements (and while, for, ...)



My question ...

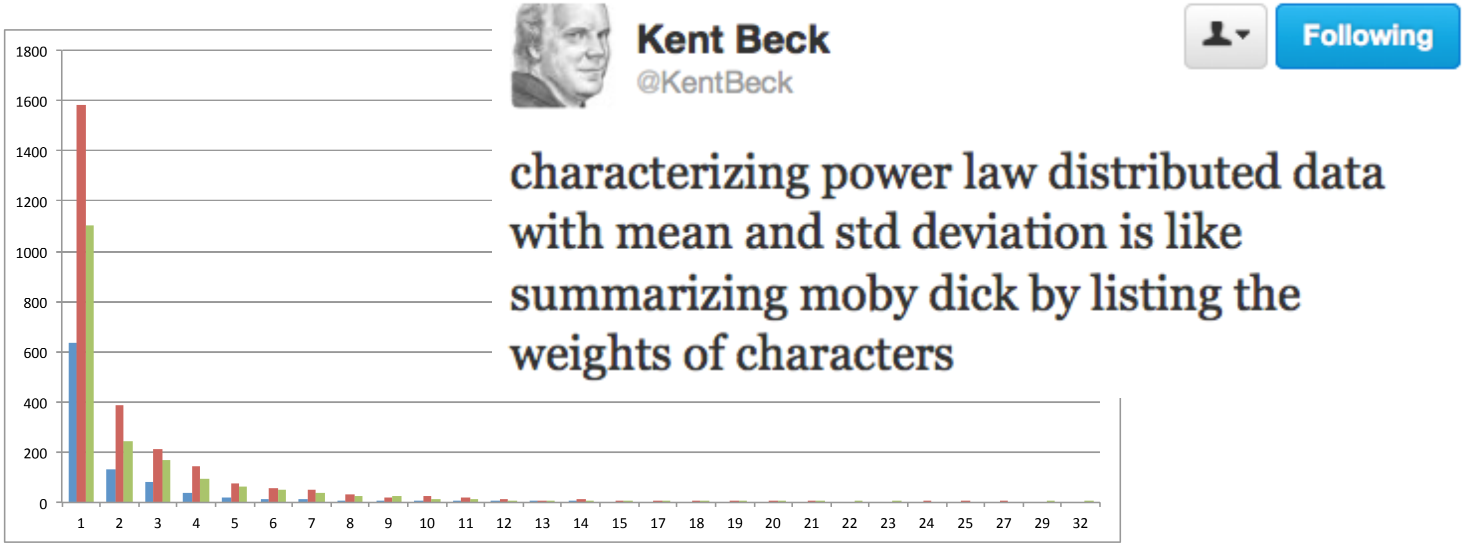
How can we aggregate this?

Option 1: Summing

	Crawljax	GOAL	Checkstyle	Springframework
Total McCabe	1814	6560	4611	22937
Total LOC	6972	25312	15994	79474
Ratio	0,260	0,259	0,288	0,288

Option 2: Average

	Crawljax	GOAL	Checkstyle	Springframework
Average McCabe	1,87	2,45	2,46	1,99



Kent Beck
@KentBeck

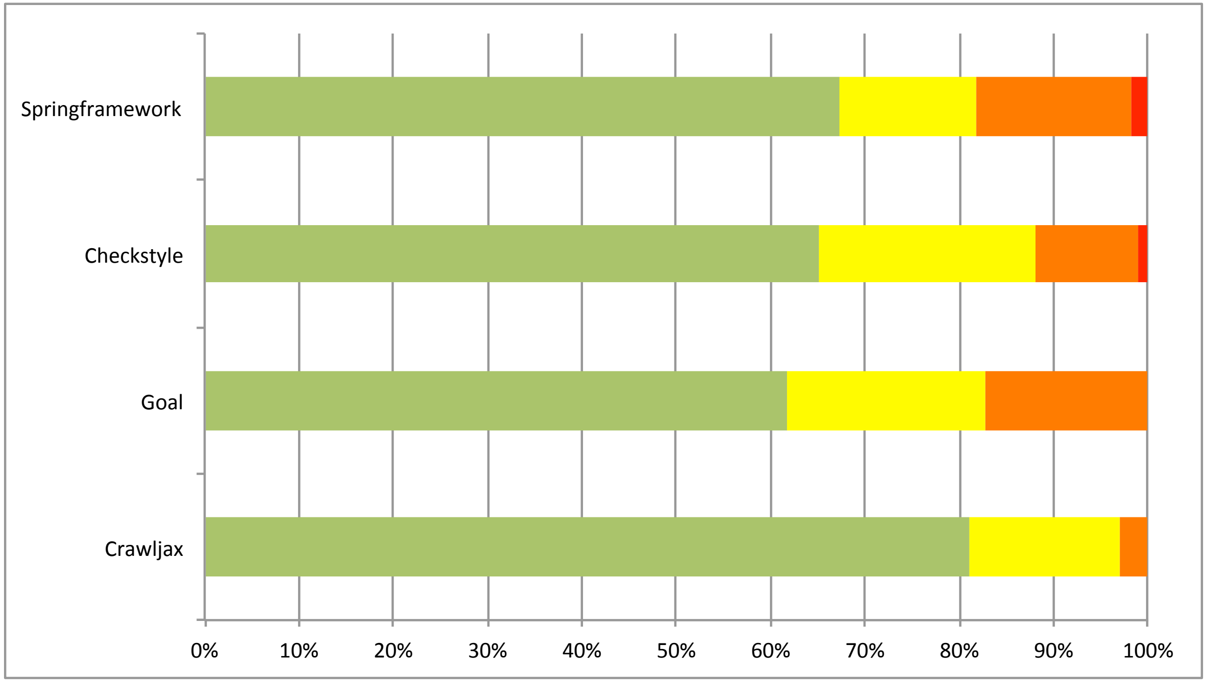
characterizing power law distributed data with mean and std deviation is like summarizing moby dick by listing the weights of characters

Option 3: quality profile

Cyclomatic complexity	Risk category
1 - 5	Low
6 - 10	Moderate
11 - 25	High
> 25	Very high

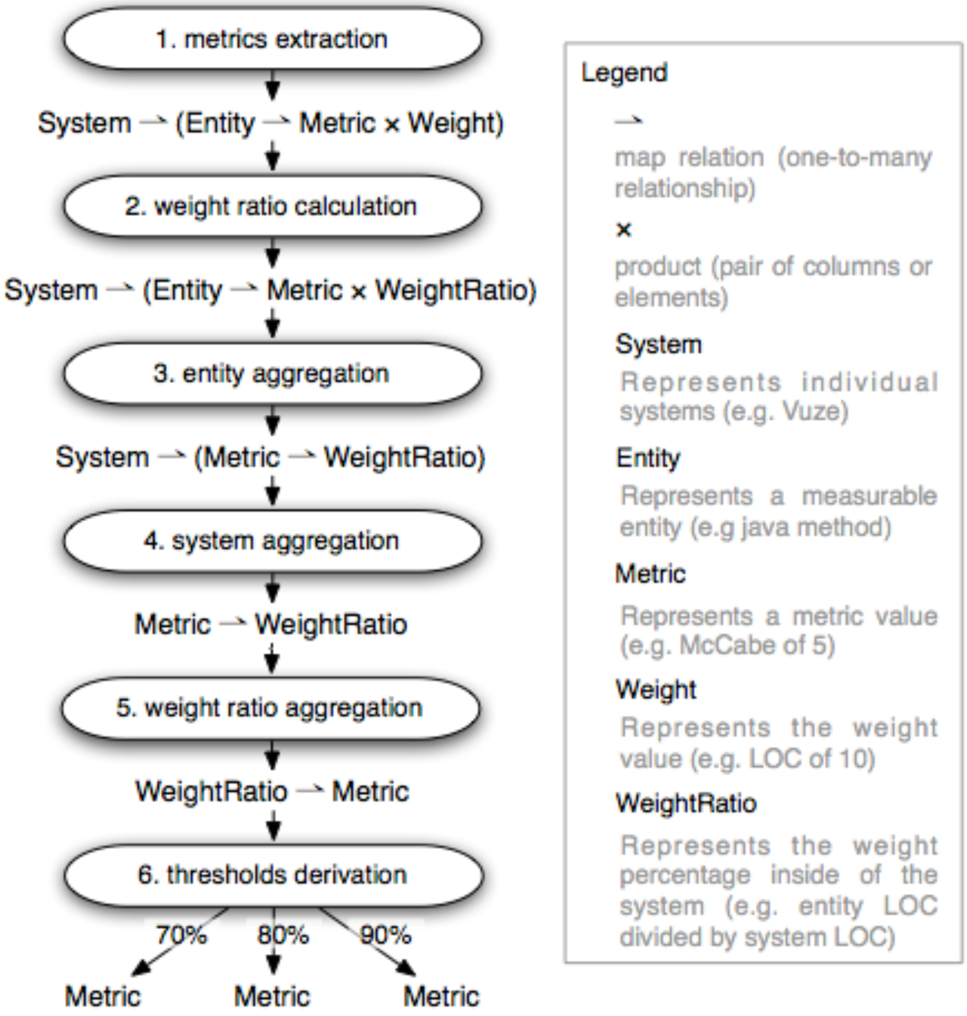
Sum lines of code per category →

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



First Level Calibration

The formal six step proces



1. metrics extraction

System → (Entity → Metric × Weight)

2. weight ratio calculation

System → (Entity → Metric × WeightRatio)

3. entity aggregation

System → (Metric → WeightRatio)

4. system aggregation

Metric → WeightRatio

5. weight ratio aggregation

WeightRatio → Metric

6. thresholds derivation

70% 80% 90%

Metric Metric Metric

Legend

→

map relation (one-to-many relationship)

×

product (pair of columns or elements)

System

Represents individual systems (e.g. Vuze)

Entity

Represents a measurable entity (e.g java method)

Metric

Represents a metric value (e.g. McCabe of 5)

Weight

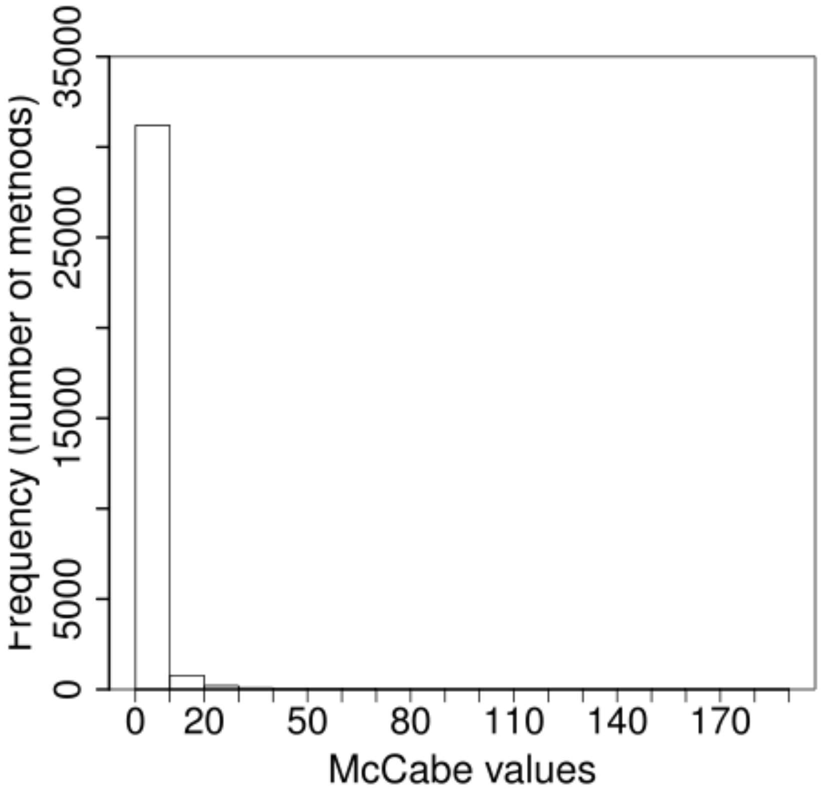
Represents the weight value (e.g. LOC of 10)

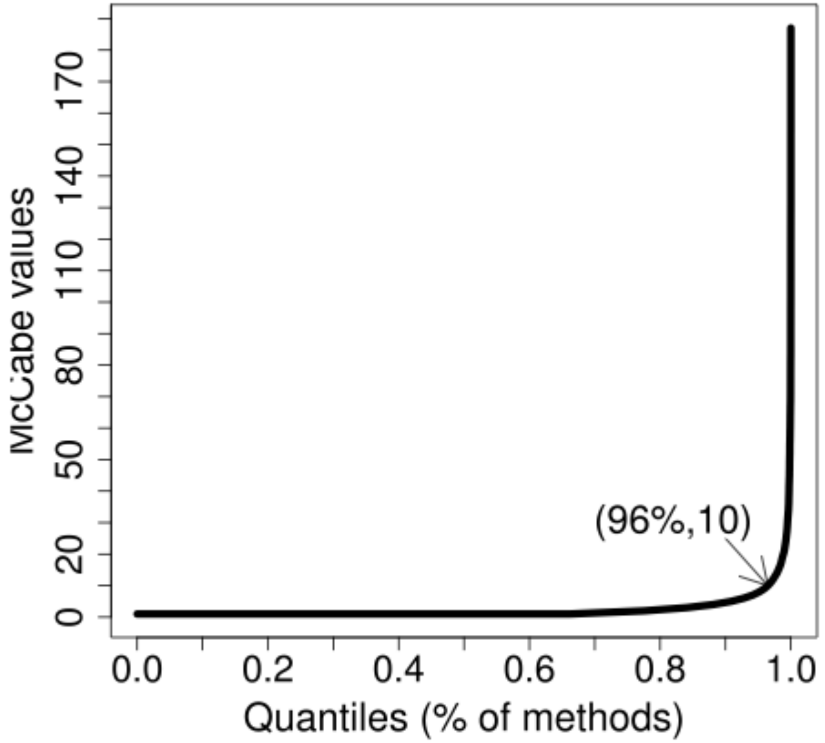
WeightRatio

Represents the weight percentage inside of the system (e.g. entity LOC divided by system LOC)

Alves, et. al., *Deriving Metric Thresholds from Benchmark Data, ICSM 2010*

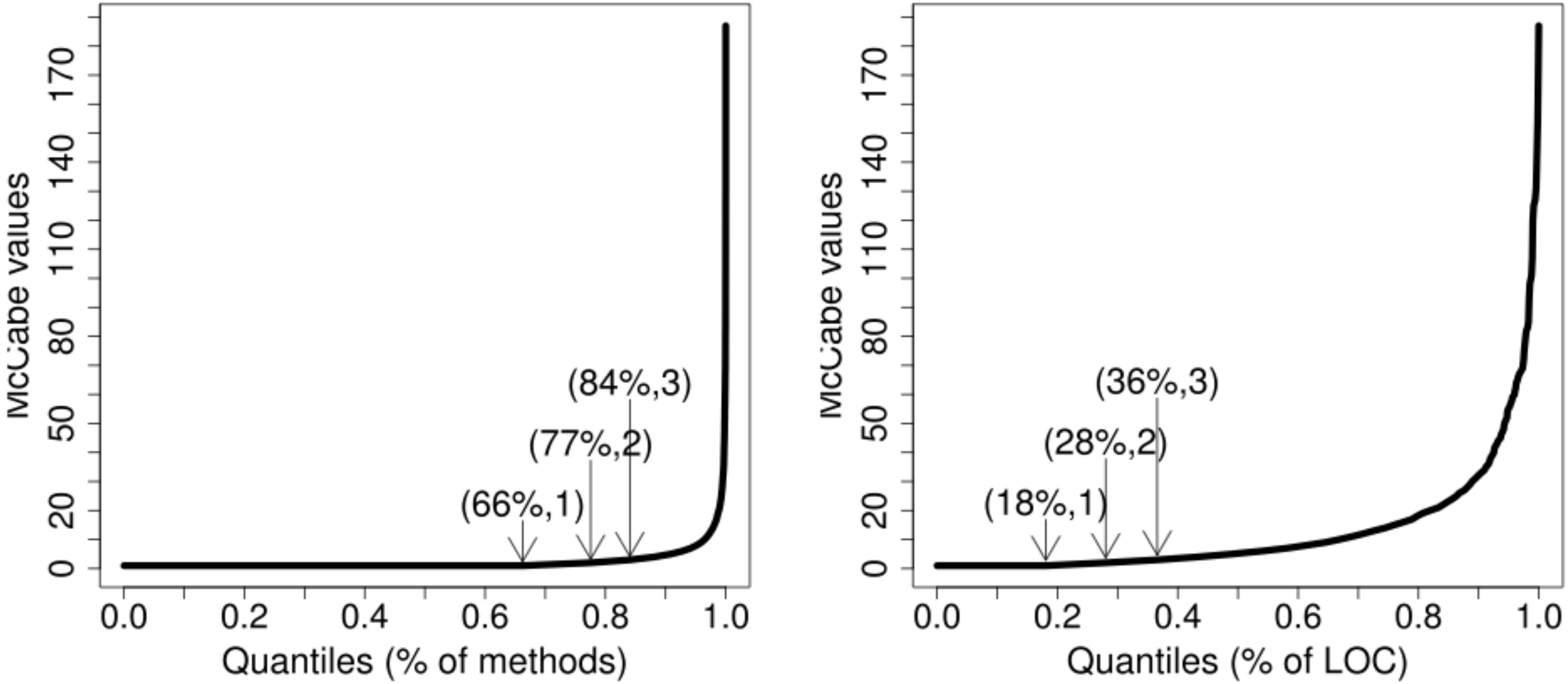
Visualizing the calculated metrics





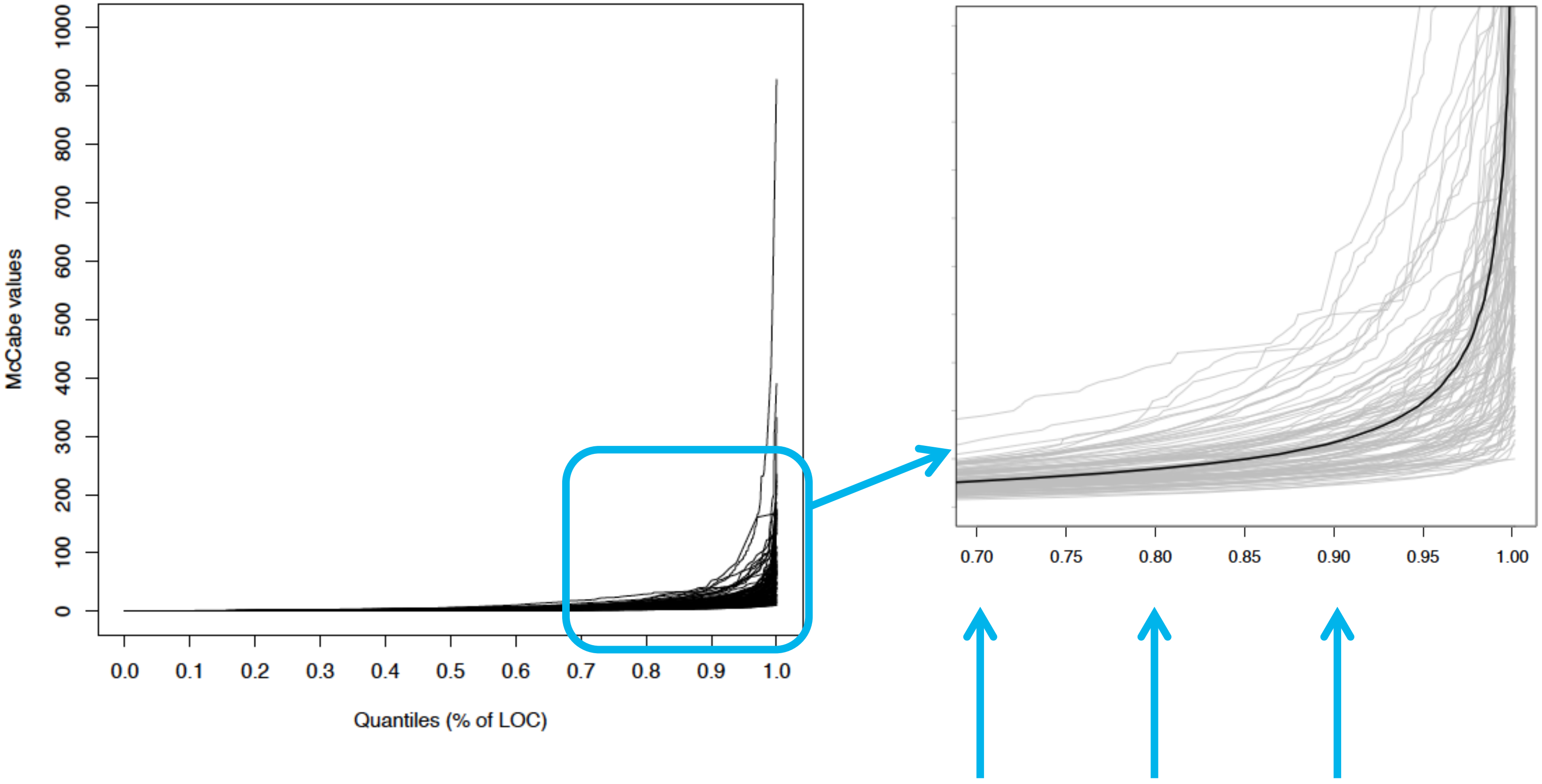
Alves, et. al., *Deriving Metric Thresholds from Benchmark Data, ICSM 2010*

Choosing a weight metric



Alves, et. al., *Deriving Metric Thresholds from Benchmark Data, ICSM 2010*

Calculate for a benchmark of systems

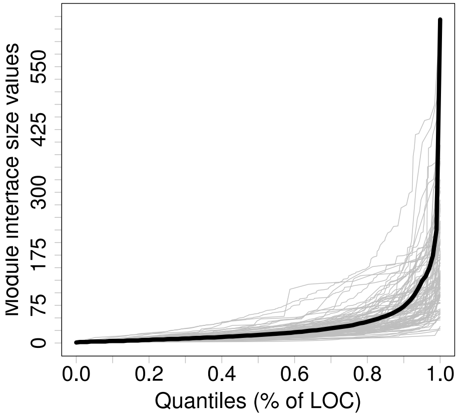


70% 80% 90%

Alves, et. al., *Deriving Metric Thresholds from Benchmark Data, ICSM 2010*

SIG Maintainability Model
Derivation metric thresholds

1. Measure systems in benchmark
2. Summarize all measurements
3. Derive thresholds that bring out the metric's variability
4. Round the thresholds



Derive & Round

Cyclomatic complexity	Risk category
1 - 5	Low
6 - 10	Moderate
11 - 25	High
> 26	Very high

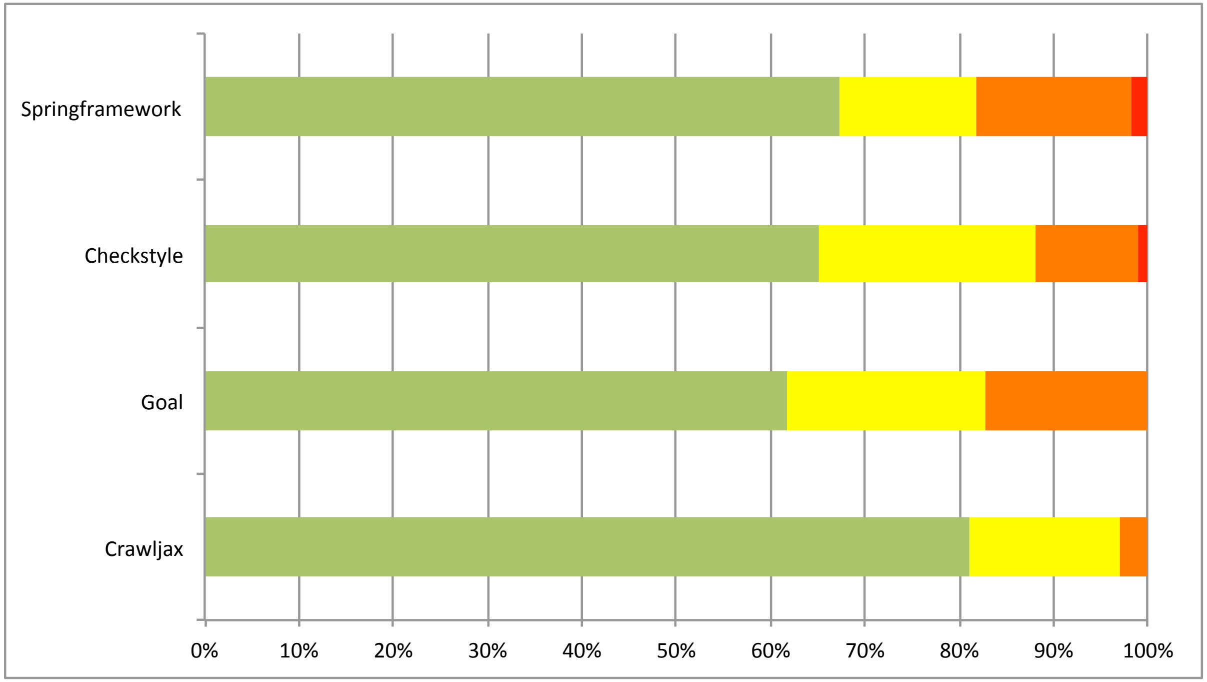
Alves, et. al., *Deriving Metric Thresholds from Benchmark Data, ICSM 2010*

The quality profile

Cyclomatic complexity	Risk category
1 - 5	Low
6 - 10	Moderate
11 - 25	High
> 25	Very high

Sum lines of code per category

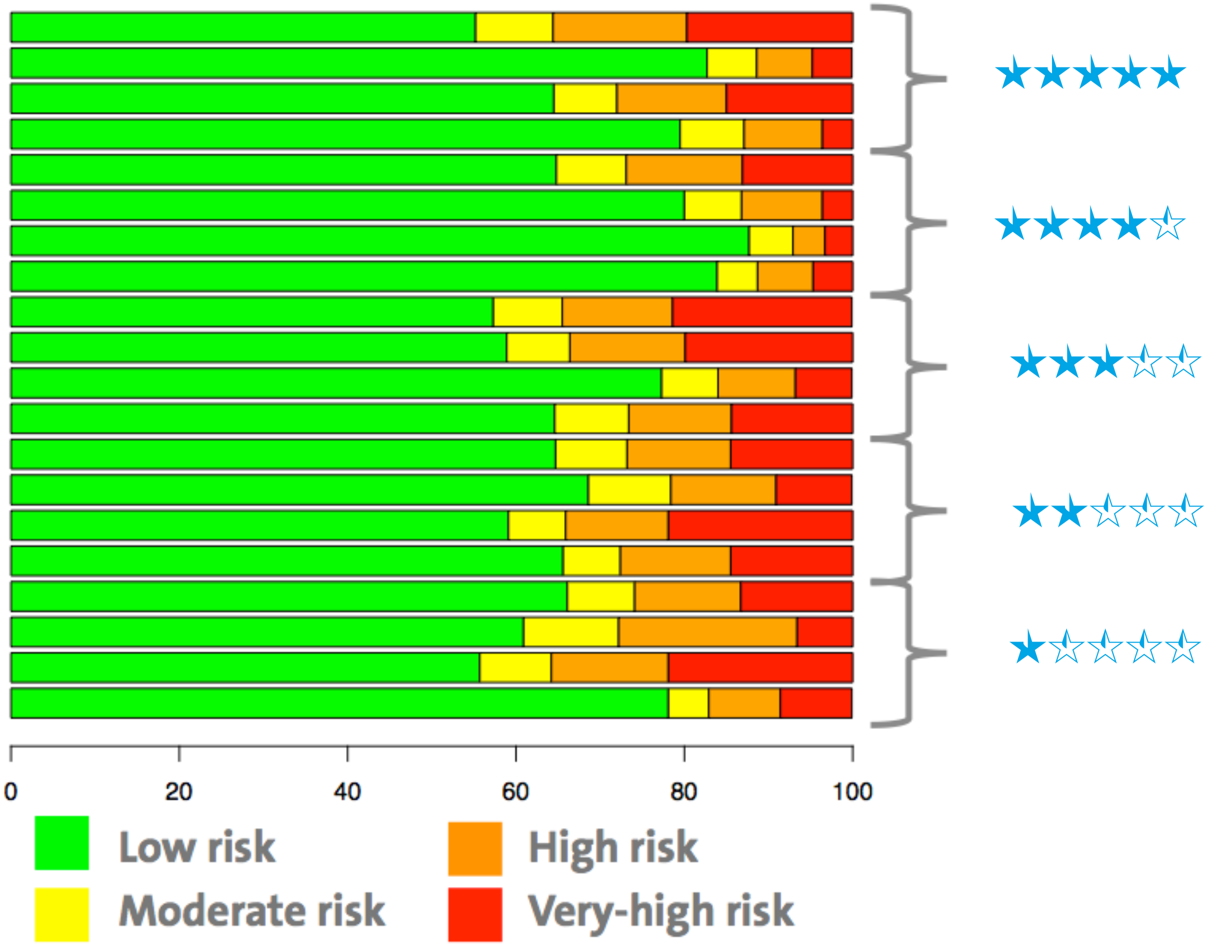
Lines of code per risk category			
Low	Moderate	High	Very high
70 %	12 %	13 %	5 %



Second Level Calibration

How to rank quality profiles?

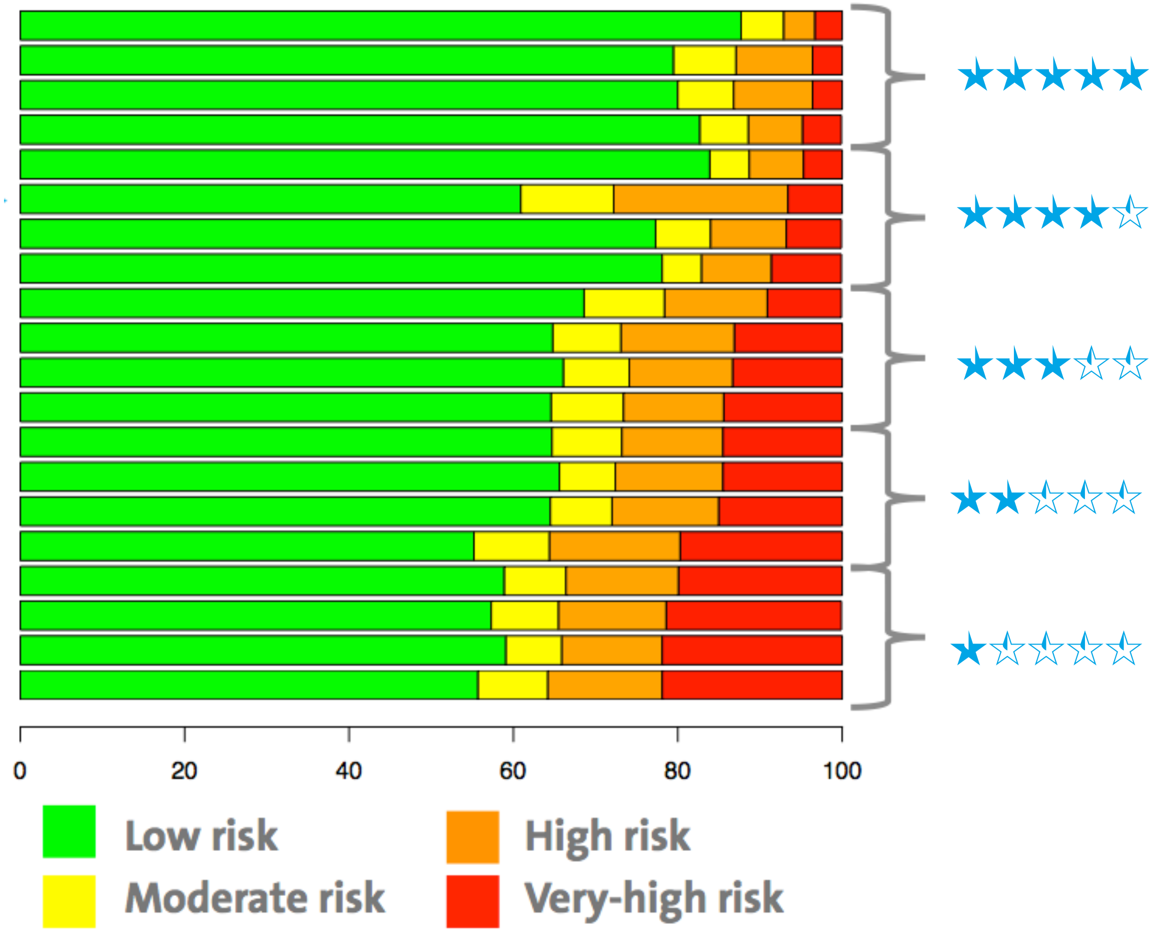
Unit Complexity profiles for 20 random systems



?

Alves, et. al., *Benchmark-based Aggregation of Metrics to Ratings*, IWSM / Mensura 2011

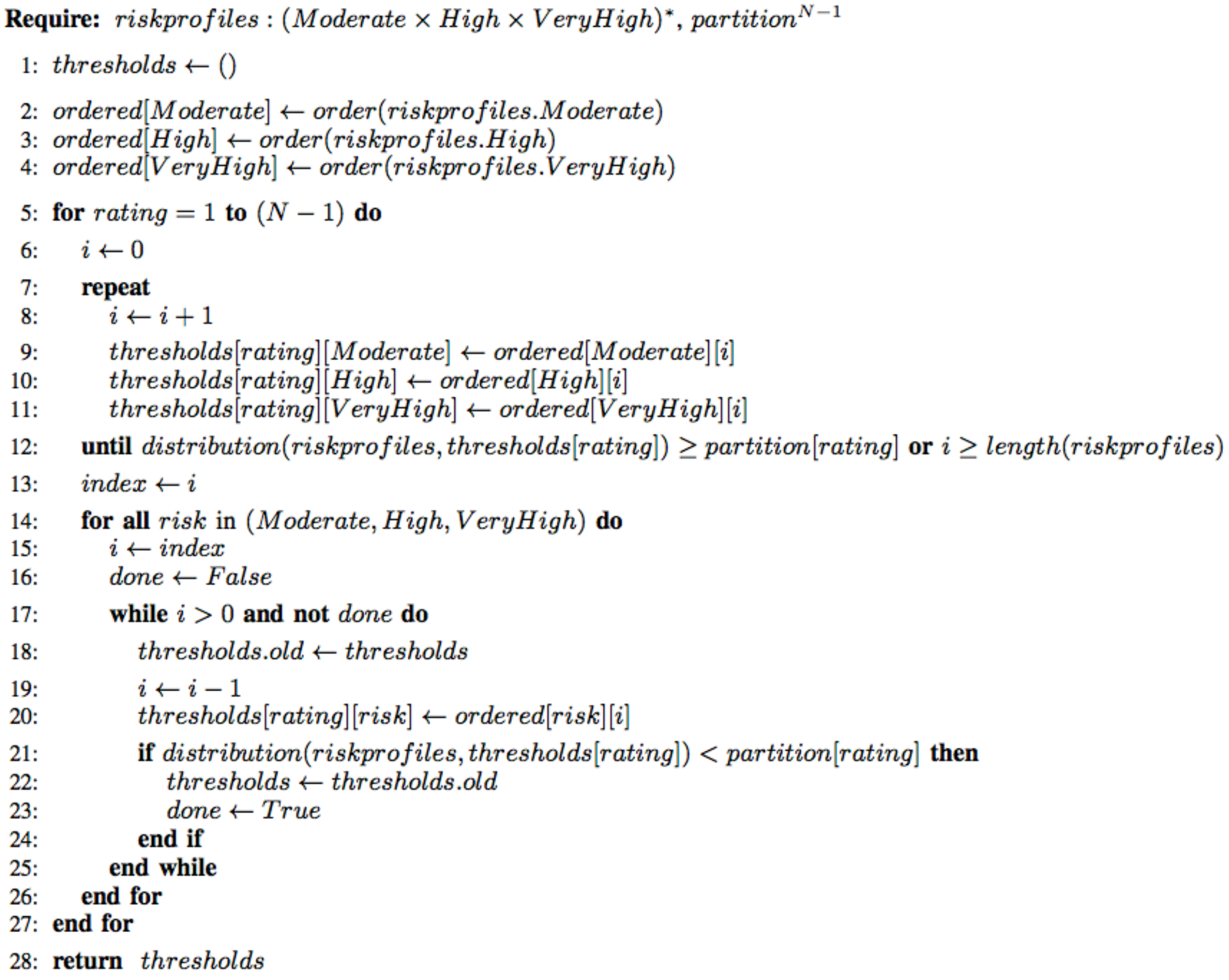
Ordering by highest-category is not enough!



?

Alves, et. al., *Benchmark-based Aggregation of Metrics to Ratings*, IWSM / Mensura 2011

A better ranking algorithm



```
Require: $riskprofiles : (Moderate \times High \times VeryHigh)^*, partition^{N-1}$
$thresholds \leftarrow ()$
$ordered[Moderate] \leftarrow order(riskprofiles.Moderate)$
$ordered[High] \leftarrow order(riskprofiles.High)$
$ordered[VeryHigh] \leftarrow order(riskprofiles.VeryHigh)$
for $rating = 1$ to $(N - 1)$ do
    $i \leftarrow 0$
    repeat
        $i \leftarrow i + 1$
        $thresholds[rating][Moderate] \leftarrow ordered[Moderate][i]$
        $thresholds[rating][High] \leftarrow ordered[High][i]$
        $thresholds[rating][VeryHigh] \leftarrow ordered[VeryHigh][i]$
    until $distribution(riskprofiles, thresholds[rating]) \geq partition[rating]$ or $i \geq length(riskprofiles)$
    $index \leftarrow i$
    for all $risk$ in $(Moderate, High, VeryHigh)$ do
        $i \leftarrow index$
        $done \leftarrow False$
        while $i > 0$ and not $done$ do
            $thresholds.old \leftarrow thresholds$
            $i \leftarrow i - 1$
            $thresholds[rating][risk] \leftarrow ordered[risk][i]$
            if $distribution(riskprofiles, thresholds[rating]) < partition[rating]$ then
                $thresholds \leftarrow thresholds.old$
                $done \leftarrow True$
            end if
        end while
    end for
end for
return $thresholds$
```

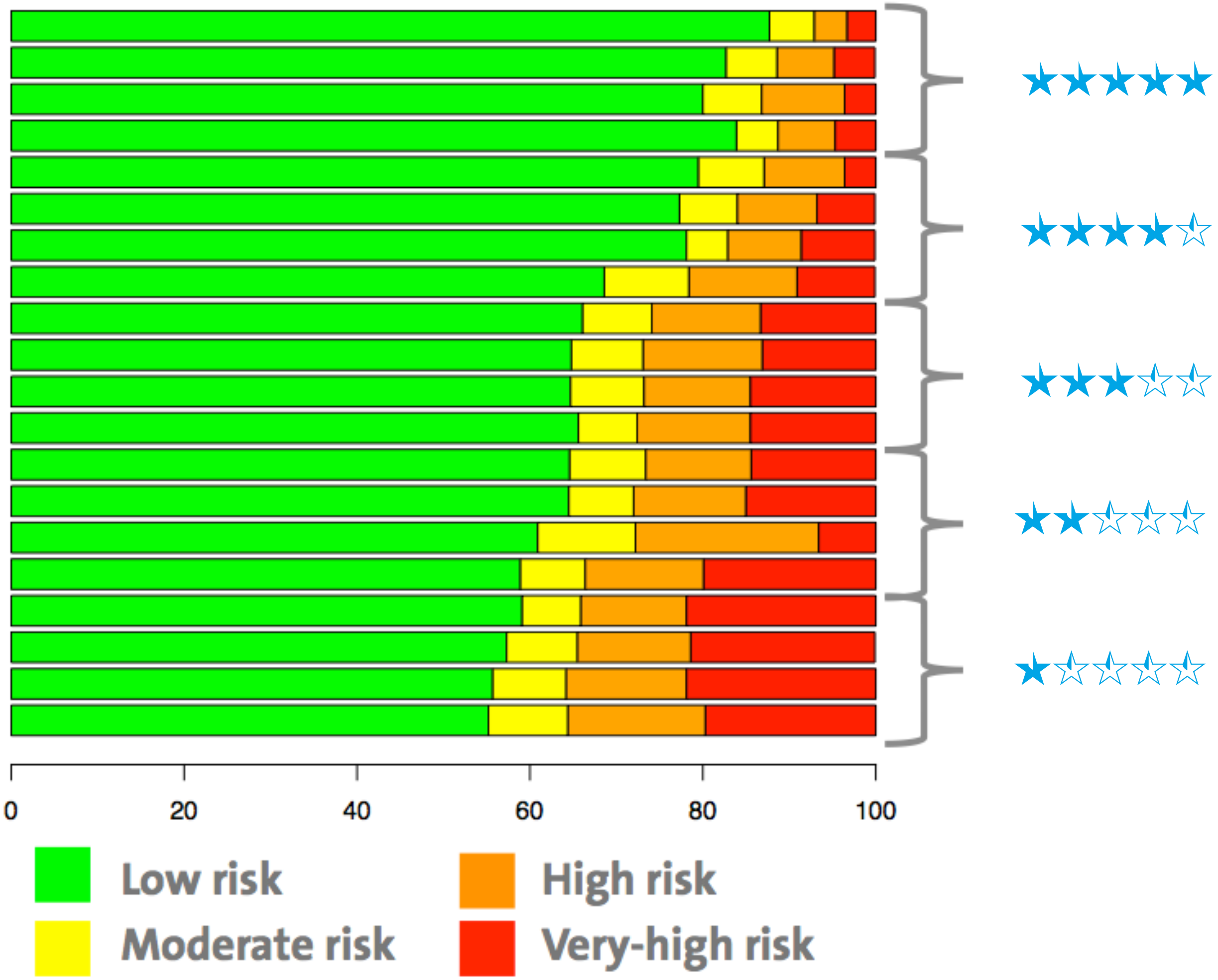
Order categories

Define thresholds of given systems

Find smallest possible thresholds

Alves, et. al., *Benchmark-based Aggregation of Metrics to Ratings*, IWSM / Mensura 2011

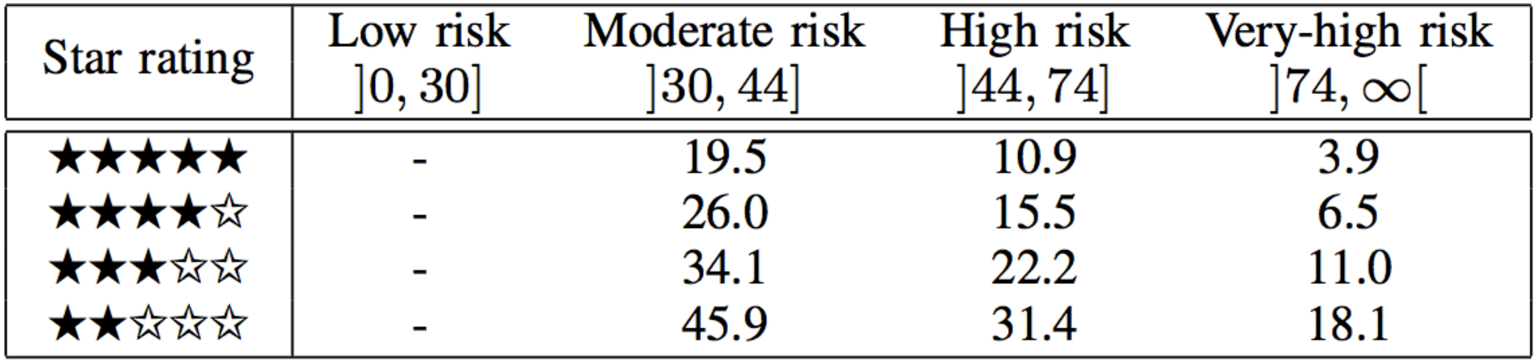
Which results in a more natural ordering



Alves, et. al., *Benchmark-based Aggregation of Metrics to Ratings*, IWSM / Mensura 2011

Second level thresholds

Unit size example



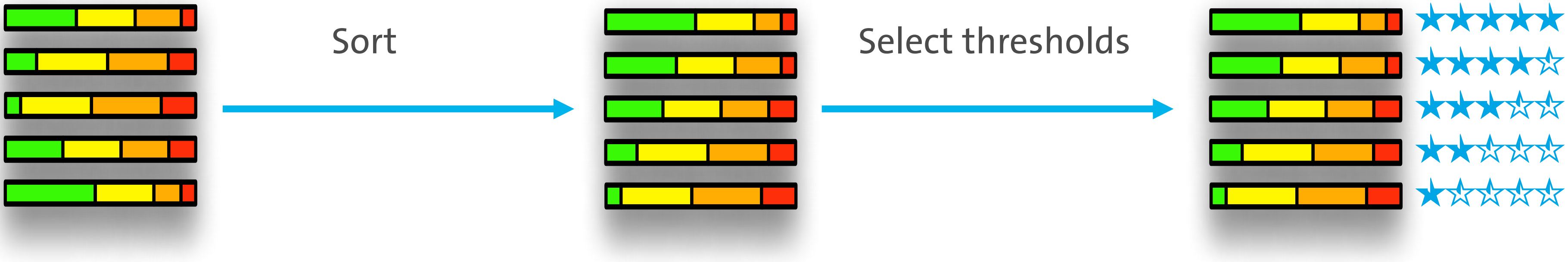
Star rating	Low risk]0, 30]	Moderate risk]30, 44]	High risk]44, 74]	Very-high risk]74, ∞[
★★★★★	-	19.5	10.9	3.9
★★★★☆	-	26.0	15.5	6.5
★★★☆☆	-	34.1	22.2	11.0
★★☆☆☆	-	45.9	31.4	18.1

Alves, et. al., *Benchmark-based Aggregation of Metrics to Ratings*, IWSM / Mensura 2011

SIG Maintainability Model

Mapping quality profiles to ratings

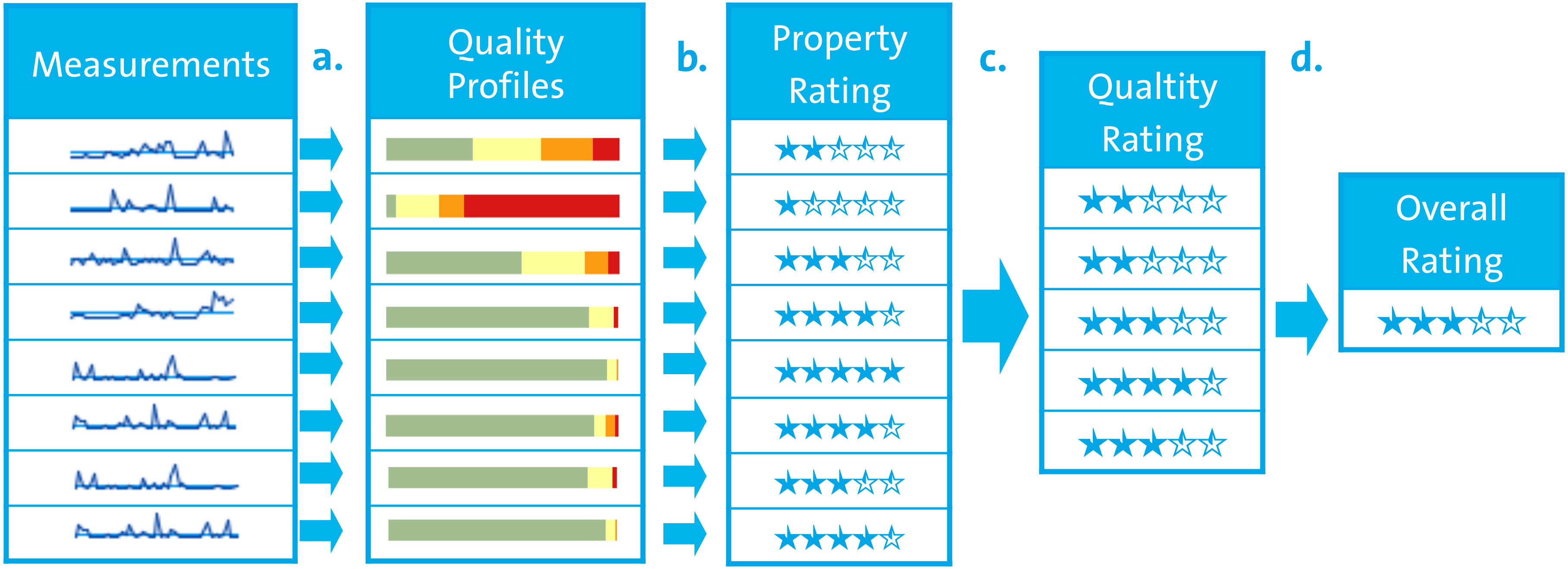
1. Calculate quality profiles for the systems in the benchmark
2. Sort quality profiles
3. Select thresholds based on 5% / 30% / 30% / 30% / 5% distribution



Alves, et. al., *Benchmark-based Aggregation of Metrics to Ratings*, IWSM / Mensura 2011

SIG measurement model

Putting it all together



Measurements

a.

Quality Profiles

b.

Property Rating

★★☆☆☆
★☆☆☆☆
★★★☆☆
★★★★☆
★★★★★
★★★★☆
★★★☆☆
★★★★☆

c.

Qualtity Rating

★★☆☆☆
★★☆☆☆
★★★☆☆
★★★★☆
★★★☆☆

d.

Overall Rating

★★★☆☆

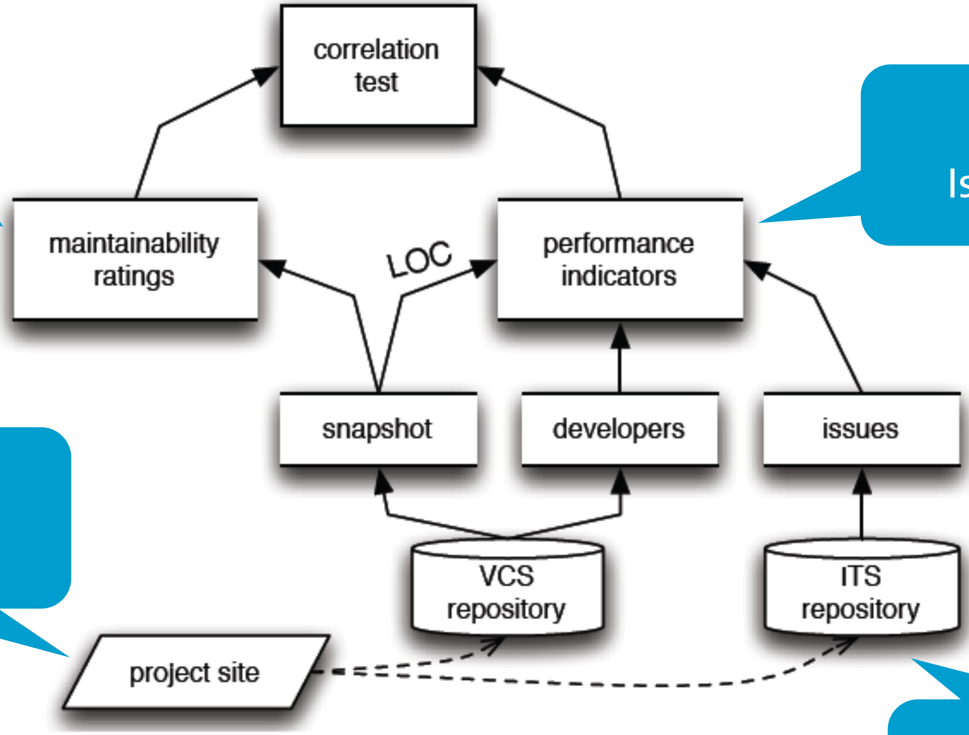
Your question ...

Does this work?

SIG Maintainability Model
Empirical validation

Internal: maintainability

external: Issue handling



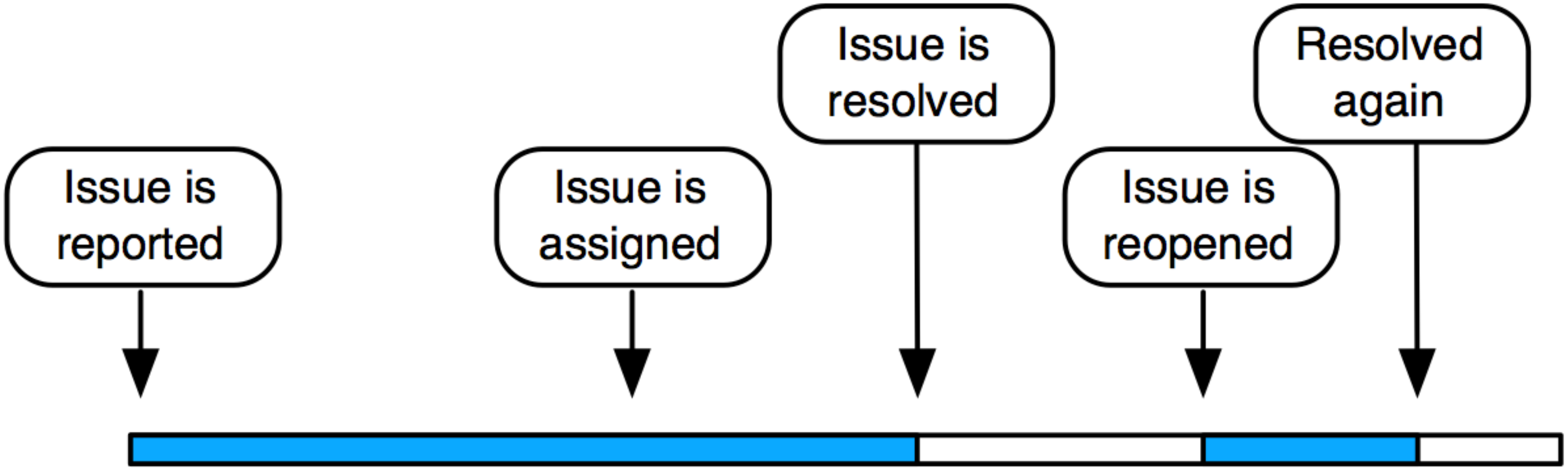
16 projects (2.5 MLOC) 150 versions

50K issues

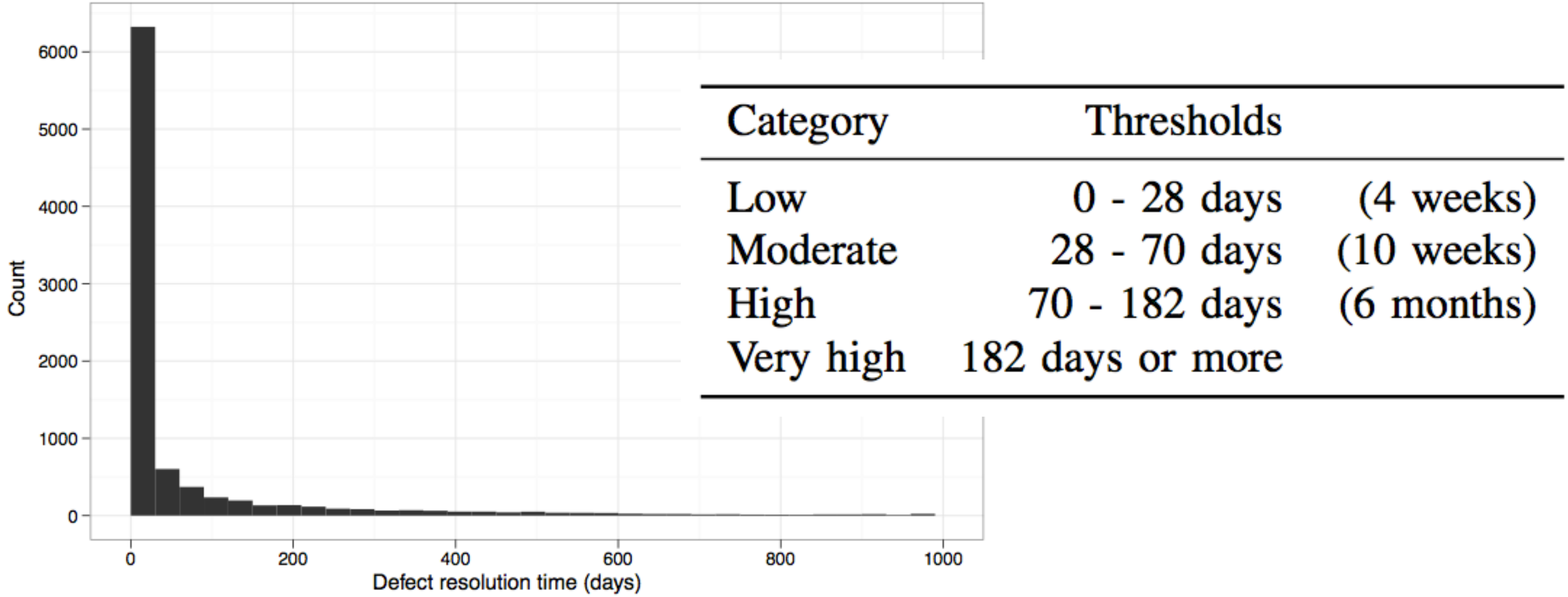
- *The Influence of Software Maintainability on Issue Handling* MSc thesis, Technical University Delft
- *Indicators of Issue Handling Efficiency and their Relation to Software Maintainability,* MSc thesis, University of Amsterdam
- *Faster Defect Resolution with Higher Technical Quality of Software, SQM 2010*

Empirical validation

The life-cycle of an issue



Empirical validation
Defect resolution time

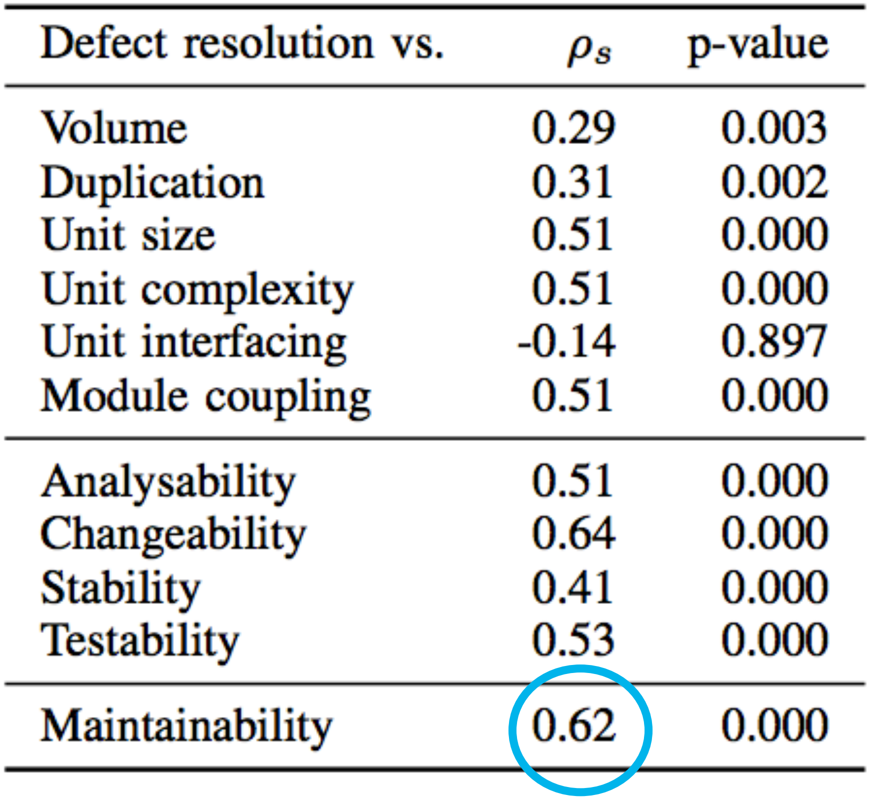


Category	Thresholds	
Low	0 - 28 days	(4 weeks)
Moderate	28 - 70 days	(10 weeks)
High	70 - 182 days	(6 months)
Very high	182 days or more	

Luijten et.al. *Faster Defect Resolution with Higher Technical Quality of Software*, SQM 2010

Empirical validation

Quantification

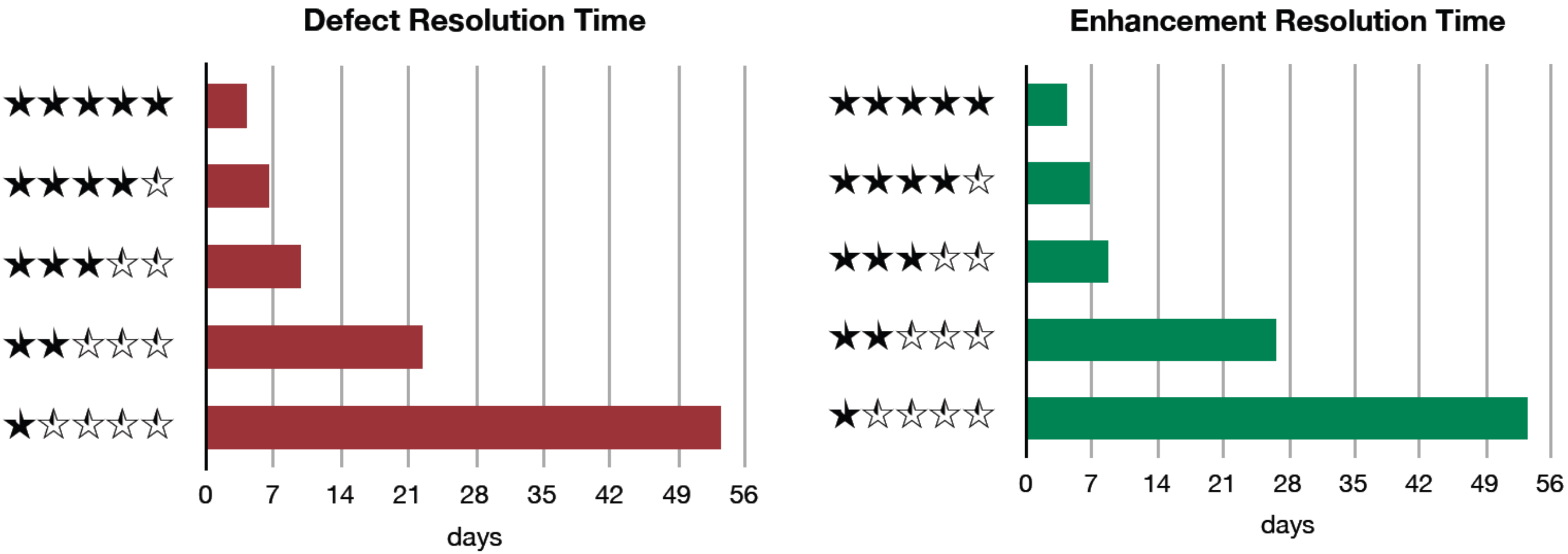


Defect resolution vs.	ρ_s	p-value
Volume	0.29	0.003
Duplication	0.31	0.002
Unit size	0.51	0.000
Unit complexity	0.51	0.000
Unit interfacing	-0.14	0.897
Module coupling	0.51	0.000
Analysability	0.51	0.000
Changeability	0.64	0.000
Stability	0.41	0.000
Testability	0.53	0.000
Maintainability	0.62	0.000

Luijten et.al. *Faster Defect Resolution with Higher Technical Quality of Software,* SQM 2010

SIG Quality Model
Quantification

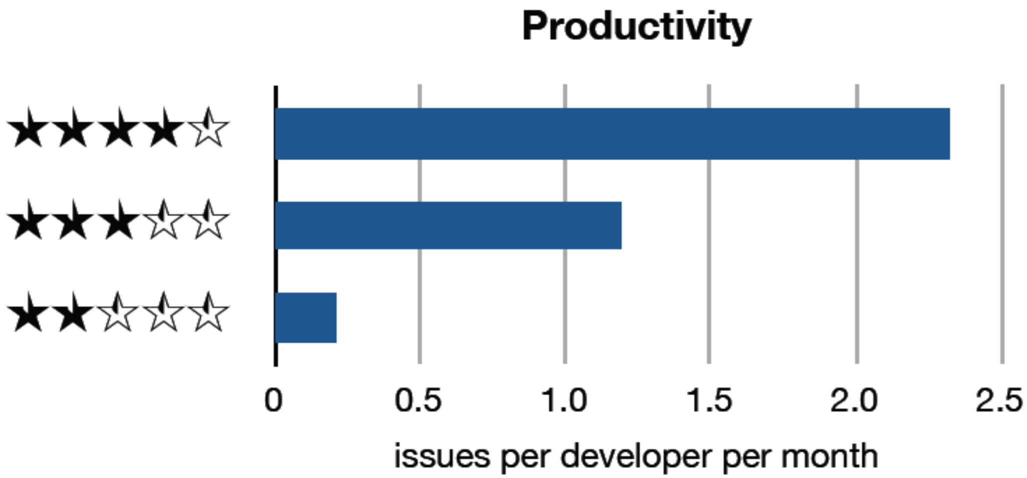
Resolution time for defects and enhancements



- Faster issue resolution with higher quality
- Between 2 stars and 4 stars, resolution speed increases by factors 3.5 and 4.0

SIG Quality Model
Quantification

Productivity (resolved issues per developer per month)



- Higher productivity with higher quality
- Between 2 stars and 4 stars, productivity increases by factor 10

Your question …

Does this work?

Yes

Theoretically

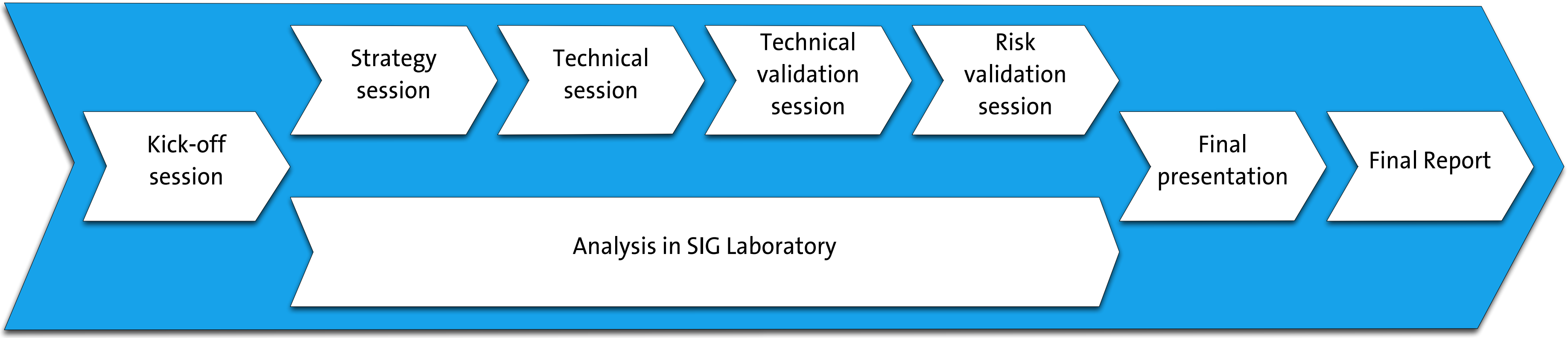
Your question ...

But is it useful?

★★★★★

Software Risk Assessment

Software Improvement Group



Example

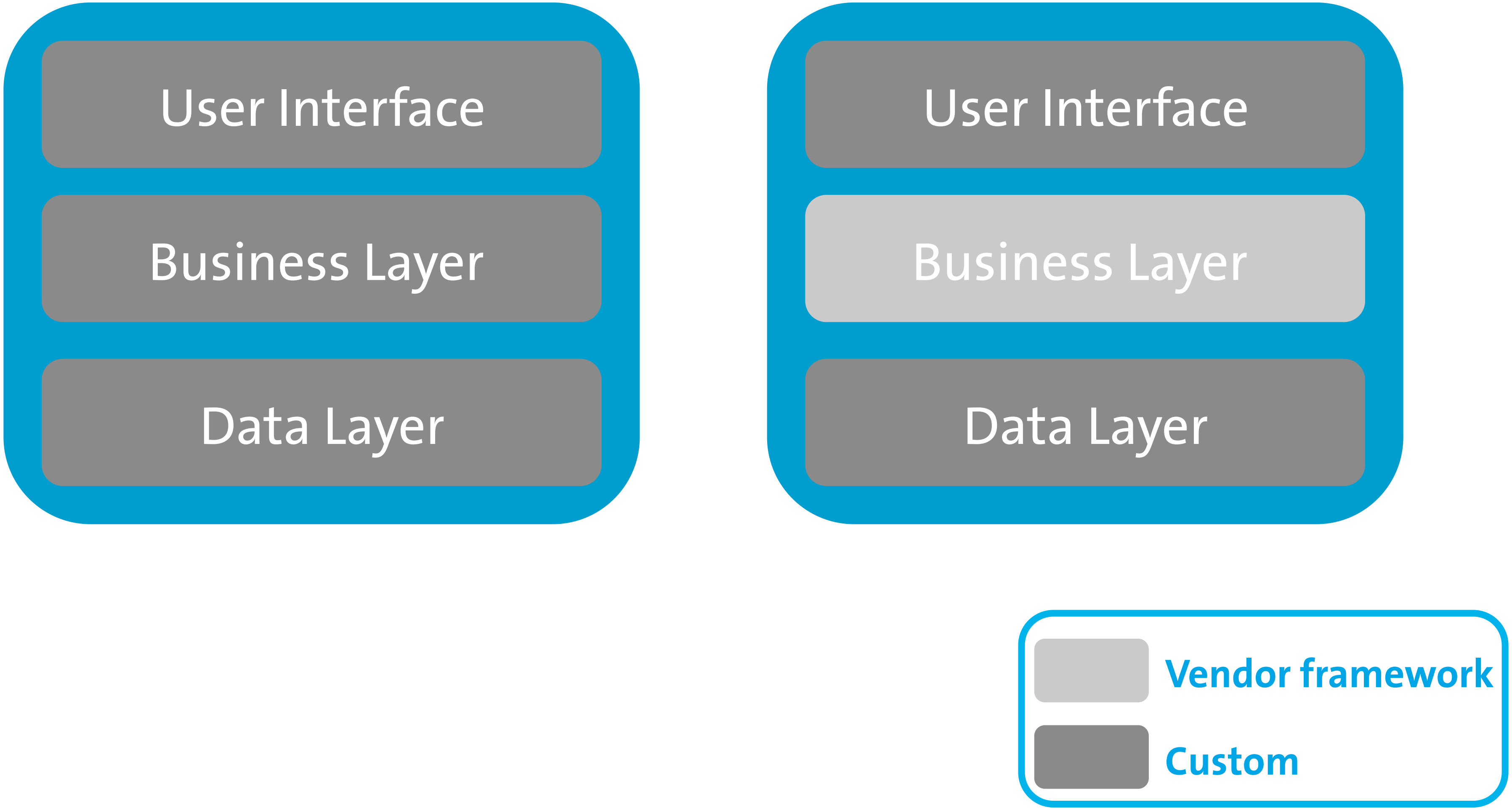
Which system to use?

★★★☆☆

★★★★☆

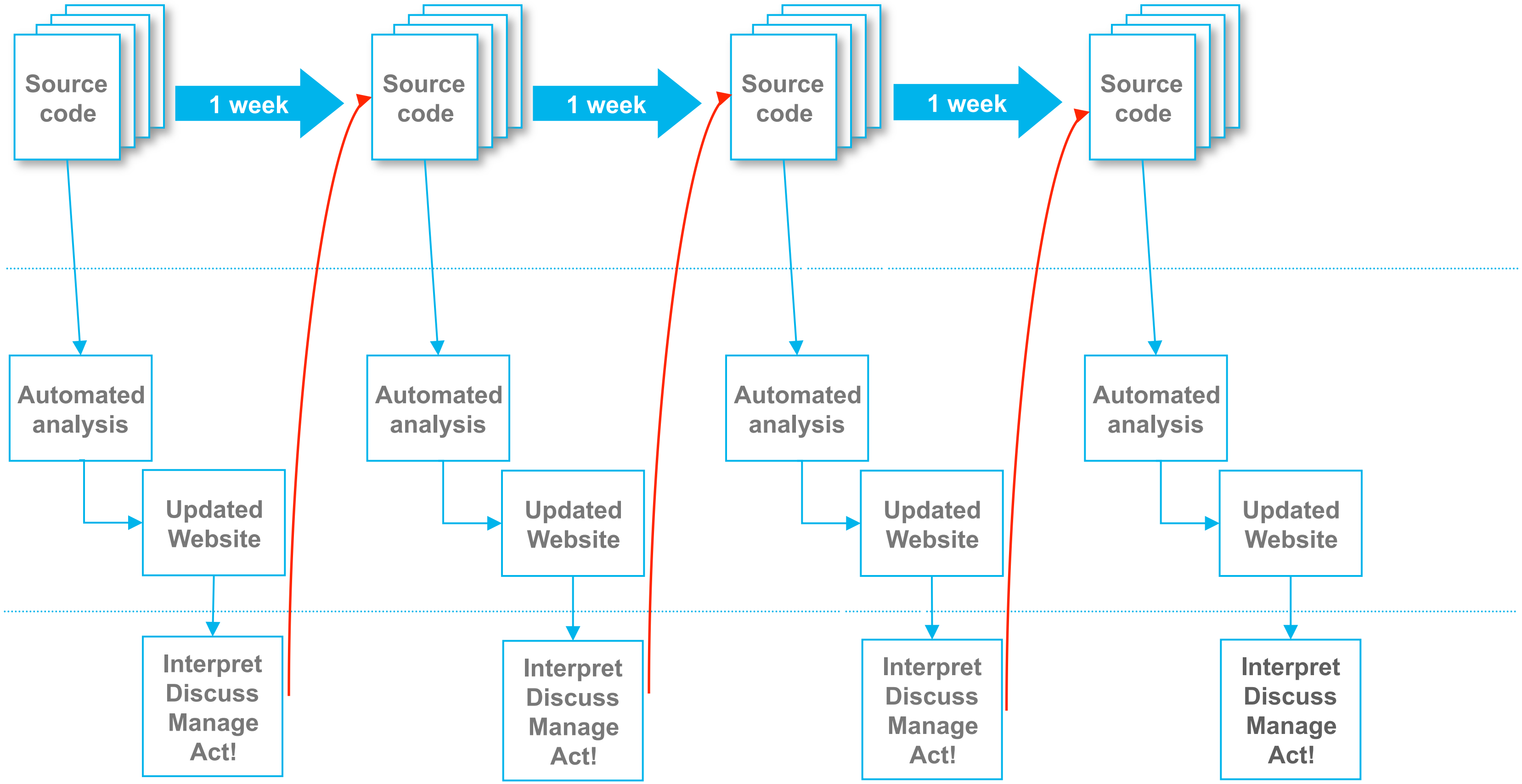
★★☆☆☆

Should we accept delay and cost overrun, or cancel the project?

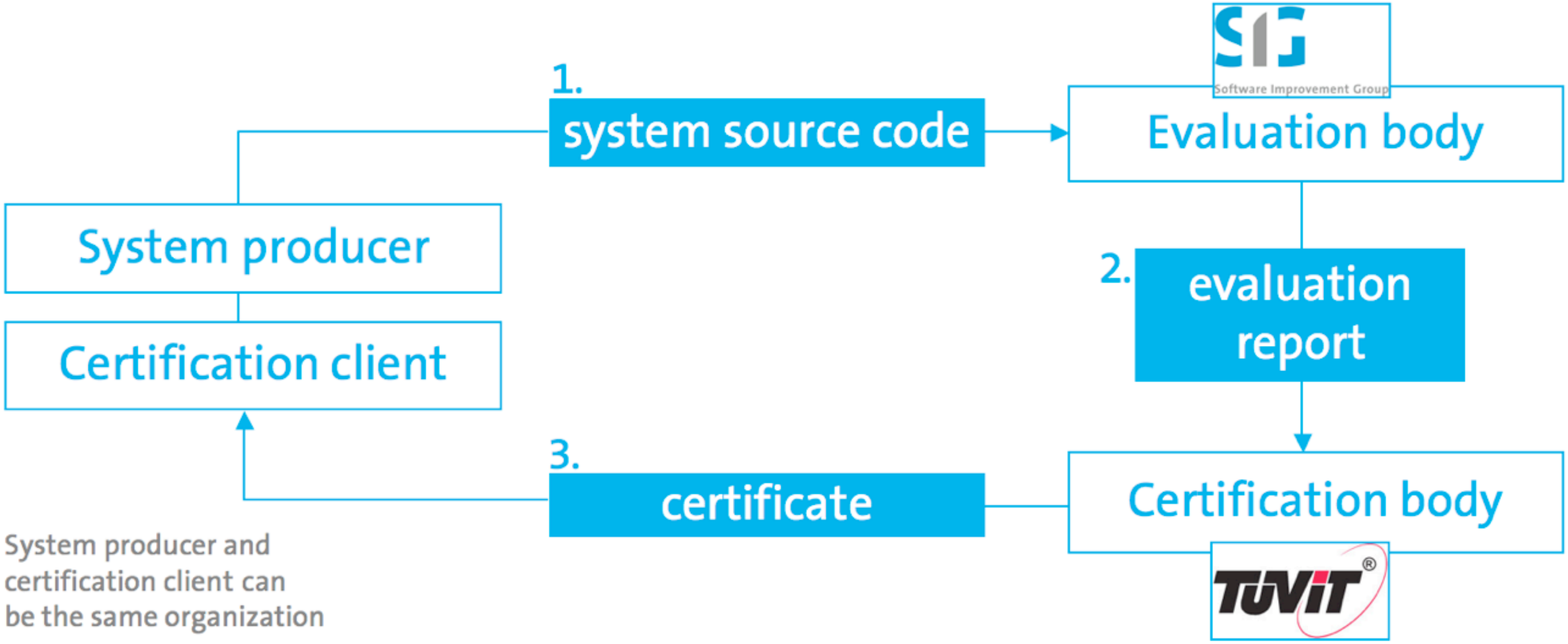


Software Monitoring

Software Improvement Group



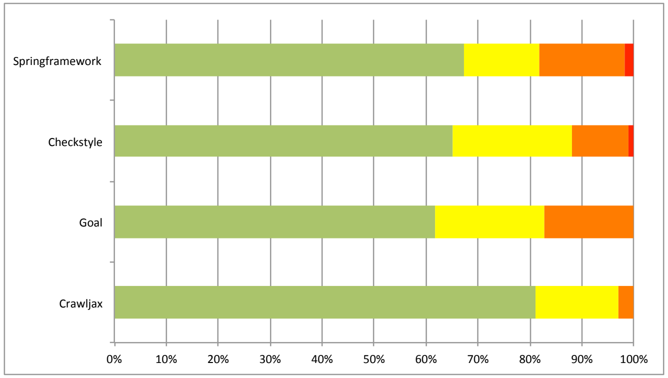
Software Product Certification

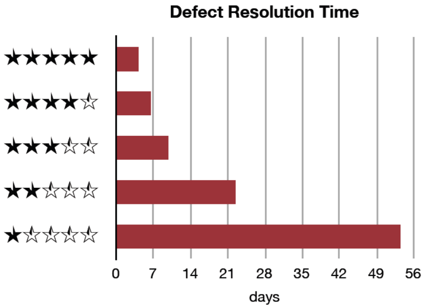


System producer and certification client can be the same organization

Summary

	Volume	Duplication	Unit size	Complexity	Unit interfacing	Module coupling	Component balance	Component indepedence
Analysability	X	X	X				X	
Modifiability		X		X		X		
Testability	X			X				X
Modularity						X	X	X
Reusability			X		X			





★★★★★

Thank you!
Eric Bouwers
e.bouwers@sig.eu