PV260 - SOFTWARE QUALITY

LECT2. Software Measurement & Metrics and their role in quality improvement

Bruno Rossi
brossi@mail.muni.cz

LAB OF SOFTWARE ARCHITECTURES
AND INFORMATION SYSTEMS

FACULTY OF INFORMATICS
MASARYK UNIVERSITY, BRNO

lasaris

Outline

- Introduction
- The Measurement Process
- Motivational Examples
- Background on Software Measurement
- The Goal Question Metrics approach
- Measures and Software Quality Improvement

  → SQALE (Software Quality Assessment Based on Lifecycle Expectations)
- Case Studies

Introduction

- The following bug (*can you spot it?*) in Apple's SSL code was undiscovered from Sept 2012 to Feb 2014 – how can it be?

FIGURE 1

The handshake algorithm containing the goto fail bug

```
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
```

FIGURE 2

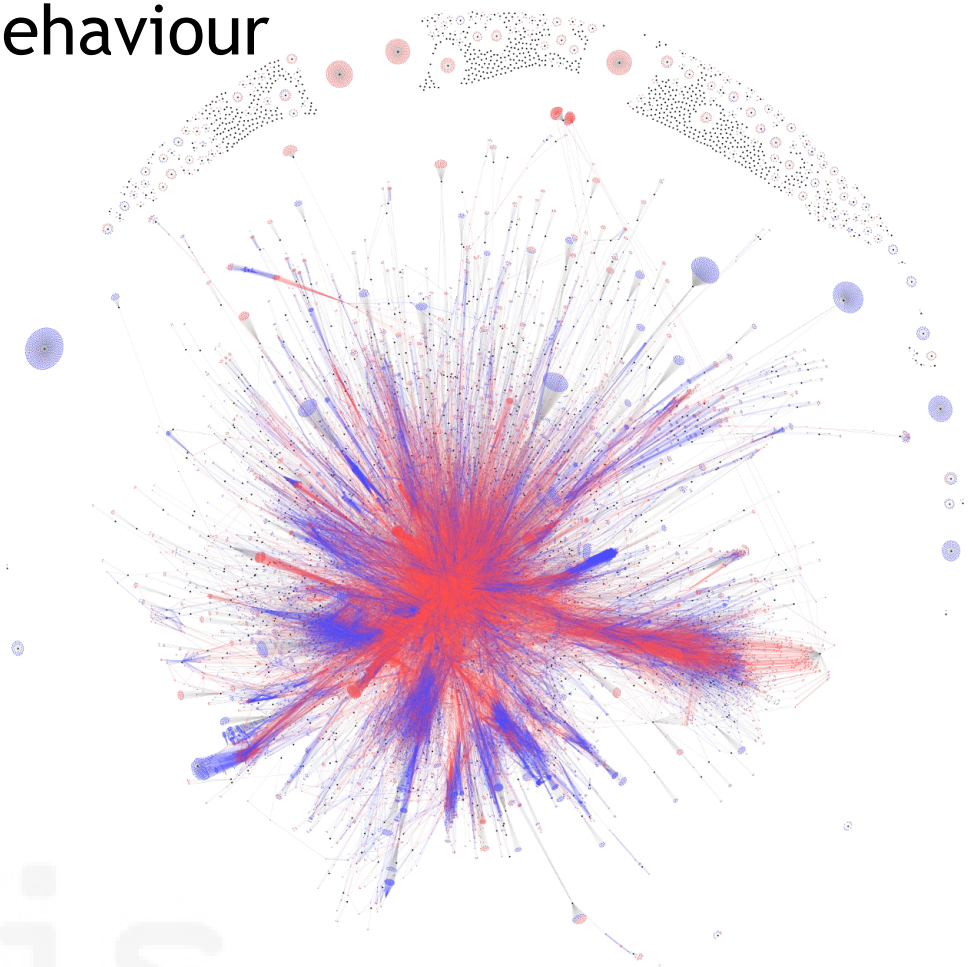
The duplicate handshake algorithm appearing immediately before the buggy block

```
if(isRsa) {
        /* … */
        if ((err = ReadyHash(&SSLHashMD5, &hashCtx)) != 0)
                goto fail;
        if ((err = SSLHashMD5.update(&hashCtx, &clientRandom)) != 0)
                goto fail;
        if ((err = SSLHashMD5.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashMD5.update(&hashCtx, &signedParams)) != 0)
                goto fail;
        if ((err = SSLHashMD5.final(&hashCtx, &hashOut)) != 0)
                goto fail;
}
```

M. Bland, "Finding more than one worm in the apple," Communications of the ACM, vol. 57, no. 7, pp. 58-64, Jul. 2014.

Introduction

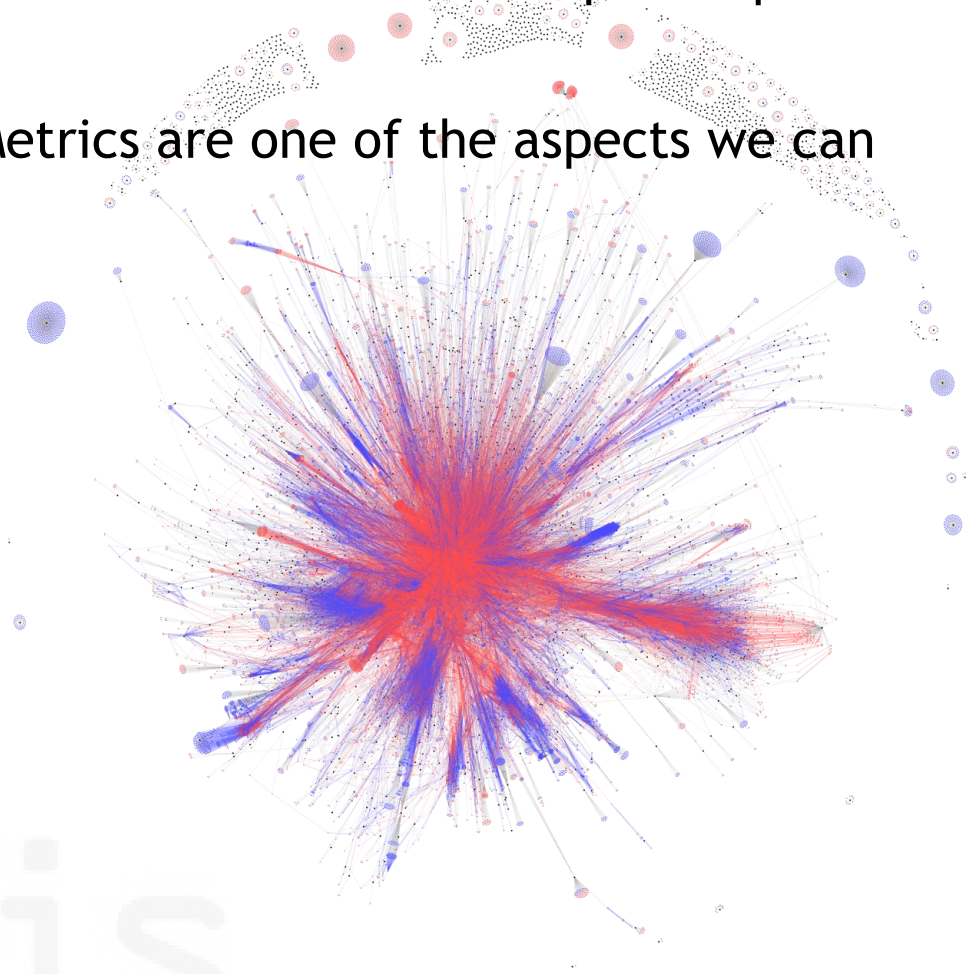
- Modern systems are very large & complex in terms of structure & runtime behaviour
- The figure on the right represents Eclipse JDT 3.5.0 (350K LOCs, 1.324 classes, 23.605 methods)



Classes → black - Methods → red - Attributes → blue. Method containment, attribute containment, and class inheritance → gray - Invocations → red - Accesses → blue

Introduction

- We need ways to understand attributes of software, represent in a concise way and use it to track for software & development process improvement
- Software Measurement and Metrics are one of the aspects we can consider



If we consider the following metrics,
what can we say?
Are they “good” metrics?

LOCs	354.780
NOM	23.605
NOC	1.324
NOP	45

LOCs=lines of code, NOM=nr. of methods
NOC=nr. of classes, NOP=nr. of packages

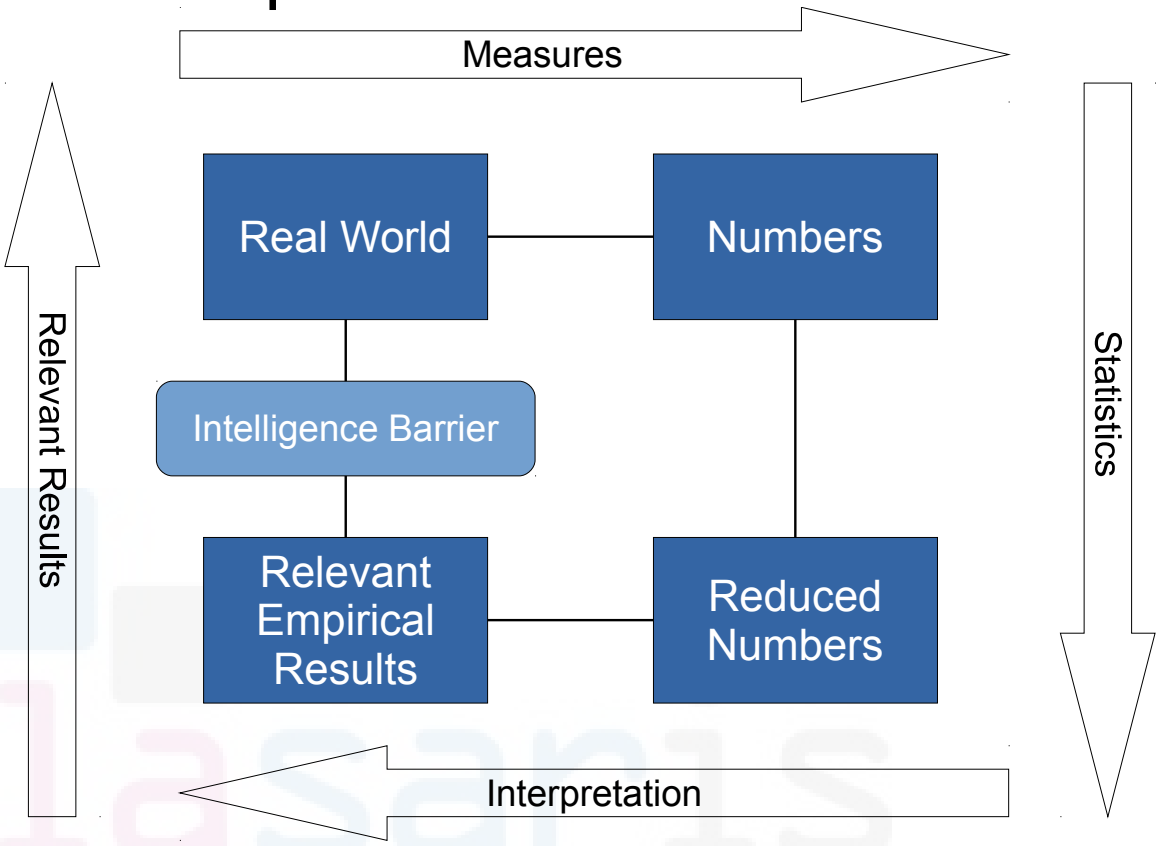
Measurement

- **Measurement** is the process by which **numbers** or **symbols** are assigned to **attributes of entities** in **the real world** in such a way as to describe them according to **clearly defined rules** (N. Fenton and S. L. Pfleeger, 1997)

→ A measurement is the **process** to define a measure

The Measurement Process

- The measurement process goes from the **real world** to the **numerical representation**
- Interpretation goes from the **numerical representation** to the **relevant empirical results**



Why Software Measurement

- To avoid anecdotal evidence without a clear study (through experiments or prototypes for example)
- To increase the visibility and the understanding of the process
- To analyze the software development
- To make predictions through statistical models

Gilbs's Principle of fuzzy targets (1988):
"Projects without clear goals will not achieve their goals clearly"

However...

- Although measurement may be integrated in development, very often objectives of measurements are not clear
- *"I measure the process because there is an automated tool that collects the metrics, but do not know how to read the data and what I can do with the data"*

Tom De Marco (1982):
"You cannot manage what you cannot measure" ...
...but you need to know what to measure and how to measure

Motivational Examples

about the pitfalls in linking the real world phenomenon to numbering systems

A Motivational Example (1/3)

- You were asked to conduct a study to evaluate whether there is discrimination among man and woman in university's enrollment
- You set-up a case study and looked at the final results

Applicants		% admitted
Men	*8442*	*44%*
Woman	*4321*	*35%*

→ Is there a discrimination in place?

→ What can you conclude from the numbers above?

A Motivational Example (2/3)

- Now look at the same study, but performed at the department level (top 6 departments):

Department	Men		Women	
	Applicants	*% admitted*	*Applicants*	*% admitted*
A	825	62%	108	82%
B	*560*	63%	25	68%
C	325	37%	593	34%
D	417	33%	375	35%
E	191	28%	393	24%
F	272	6%	341	7%

- There does not seem to be any discrimination against women! The conclusion is that **women tended to apply to more competitive departments than men**
- The effect we just saw is called Simpson's paradox

Source of the example: http://en.wikipedia.org/wiki/Simpson%27s_paradox – considering the following papers:

J. Pearl (2000). Causality: Models, Reasoning, and Inference, Cambridge University Press.

P.J. Bickel, E.A. Hammel and J.W. O'Connell (1975). "Sex Bias in Graduate Admissions: Data From Berkeley. Science 187 (4175): 398-404.

A Motivational Example (3/3)

- Simpsons' paradox: How can it be?
- It can happen that:

```
a/b < A/B

c/d < C/D

(a + c)/(b + d) > (A + C)/(B + D)
```

- e.g.

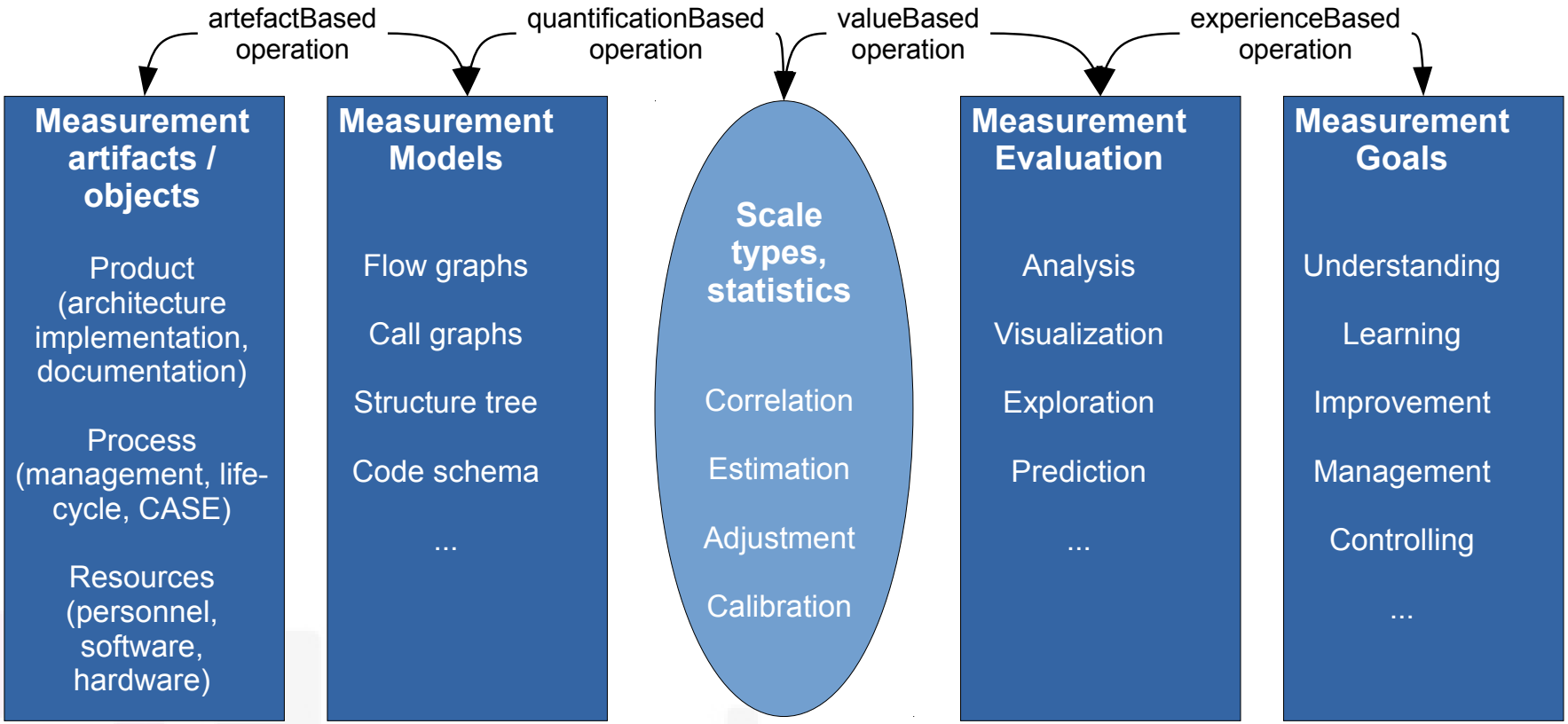
```
1/5 < 2/8

6/8 < 4/5

7/13 > 6/13
```

Dept	Men		Women	
	Applicants	*admitted*	*Applicants*	*admitted*
A	5	20%	**8**	**25%**
B	*8*	75%	**5**	**80%**
Total	**13**	**53%**	13	46%

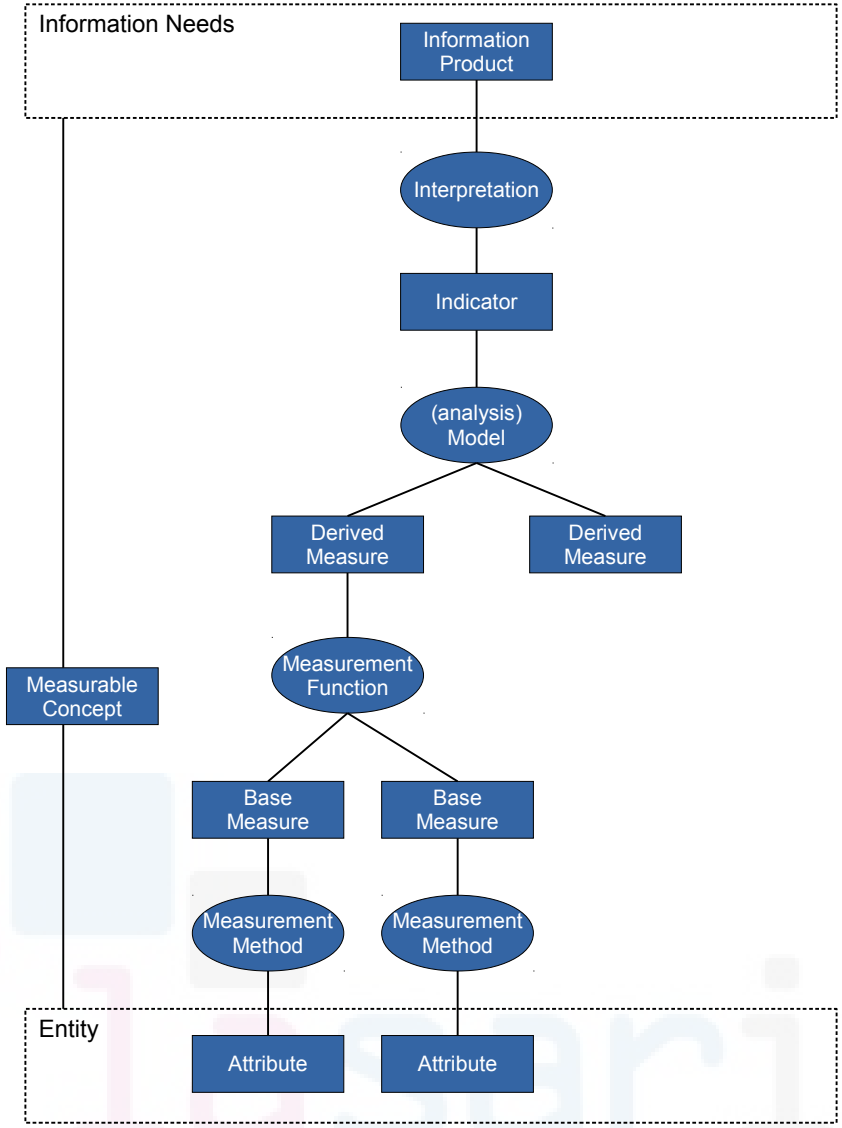
- It is the result of not considering an hidden variable, as in the example not considering the difficulty of entering a certain department

Background on Software Measurement

Software Measurement Methods



Measurement Information Model (ISO/IEC 15939)

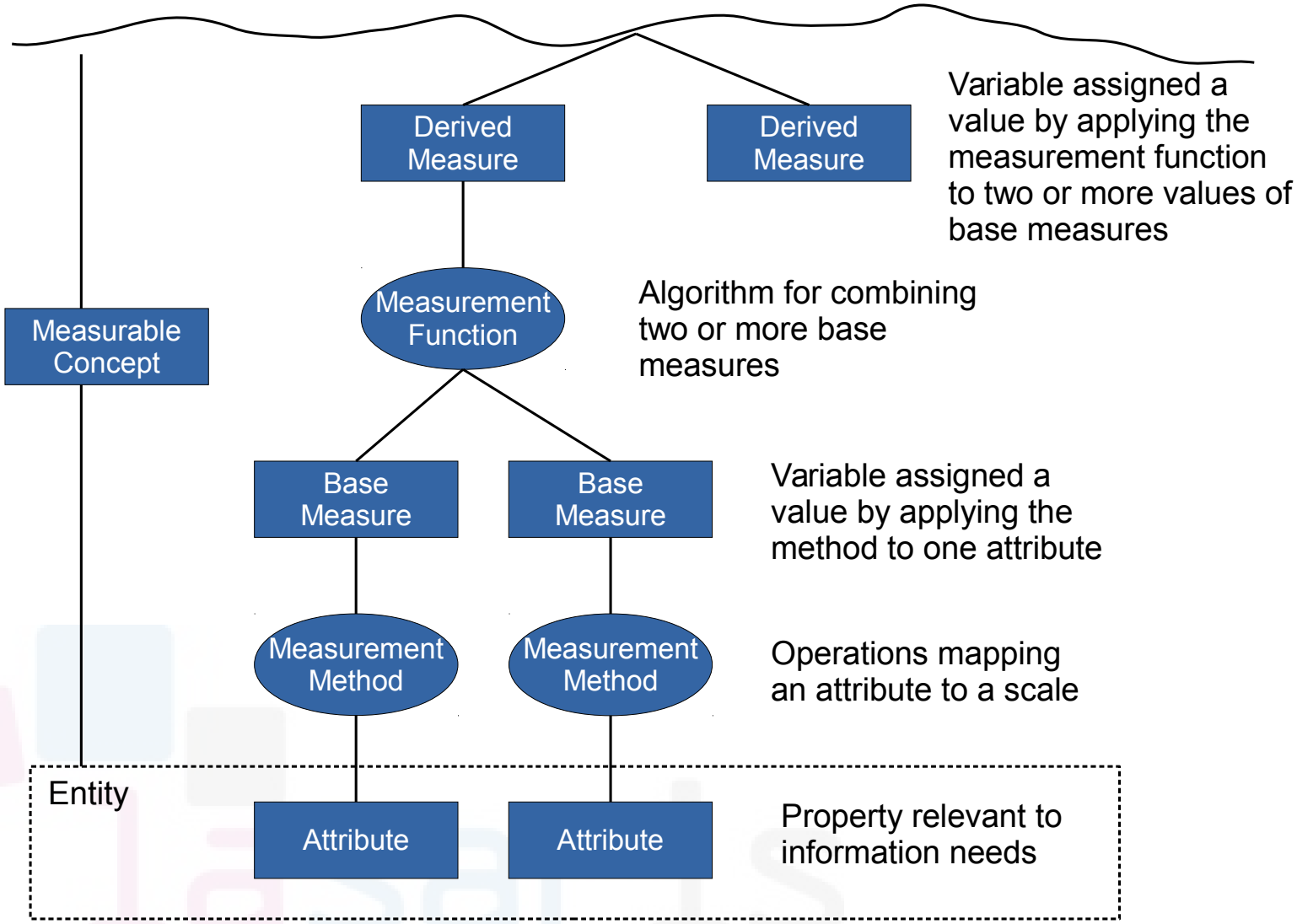


Measurable Concept:

abstract relationship between attributes of entities and information needs

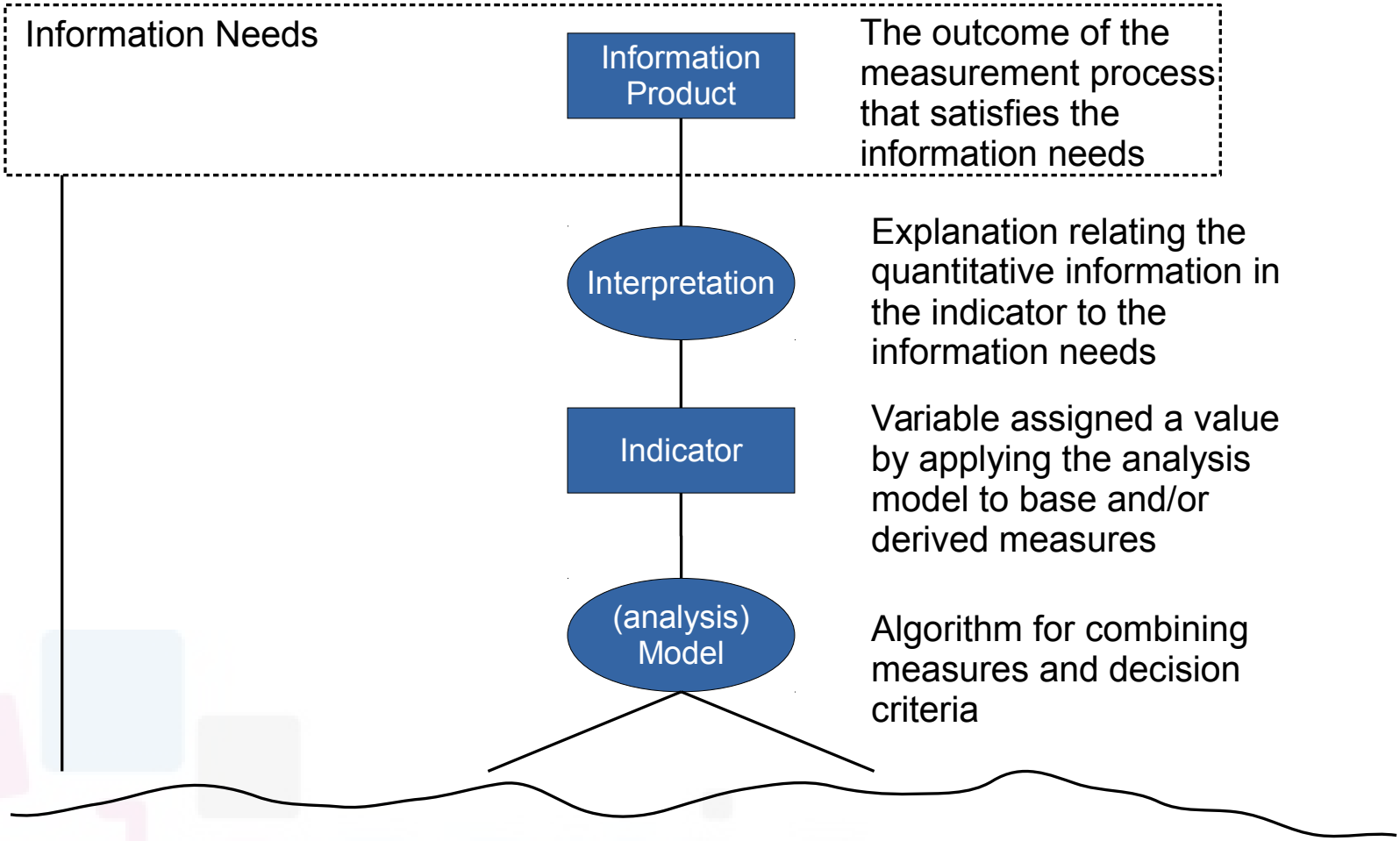
Measurement Information Model (ISO/IEC 15939)

Bottom part

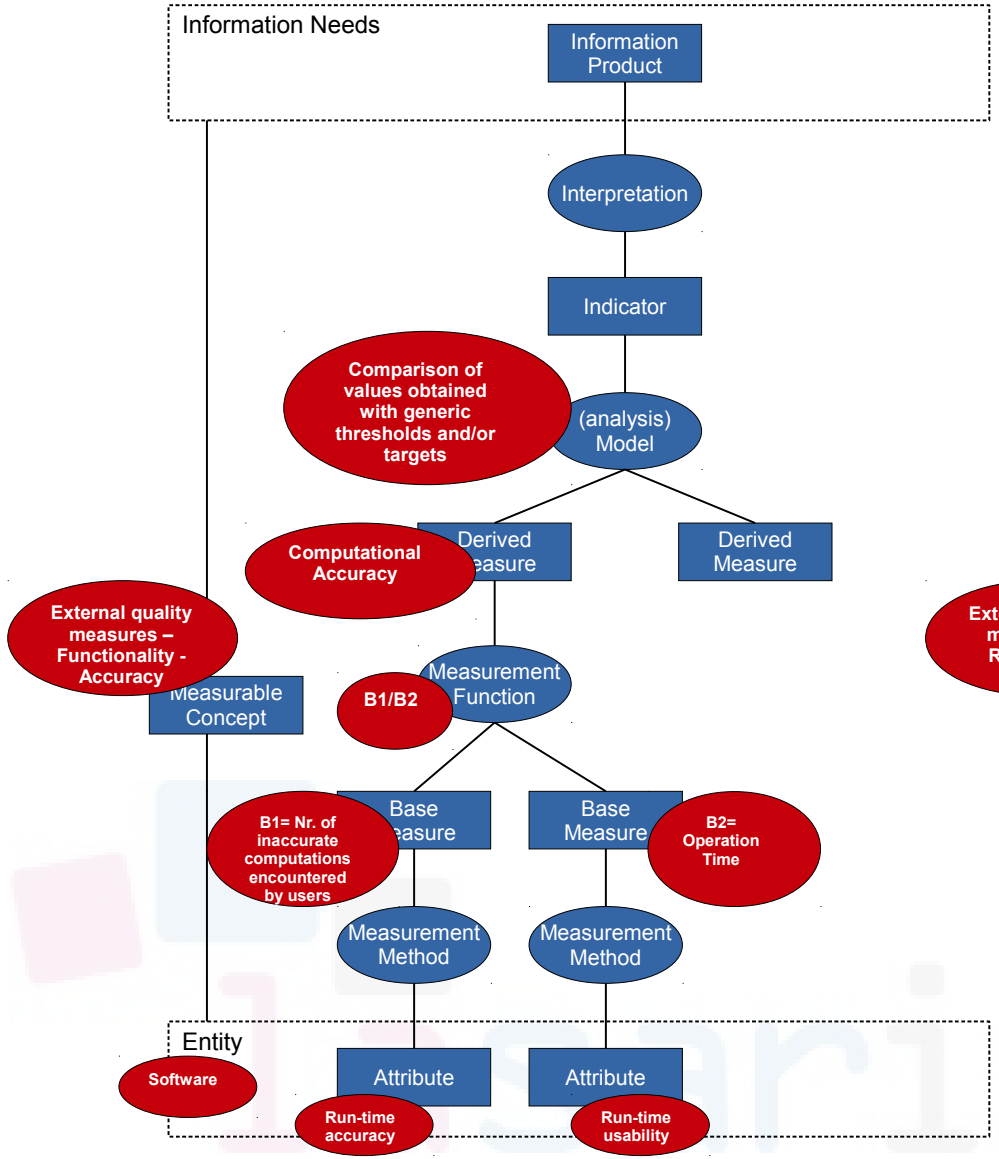


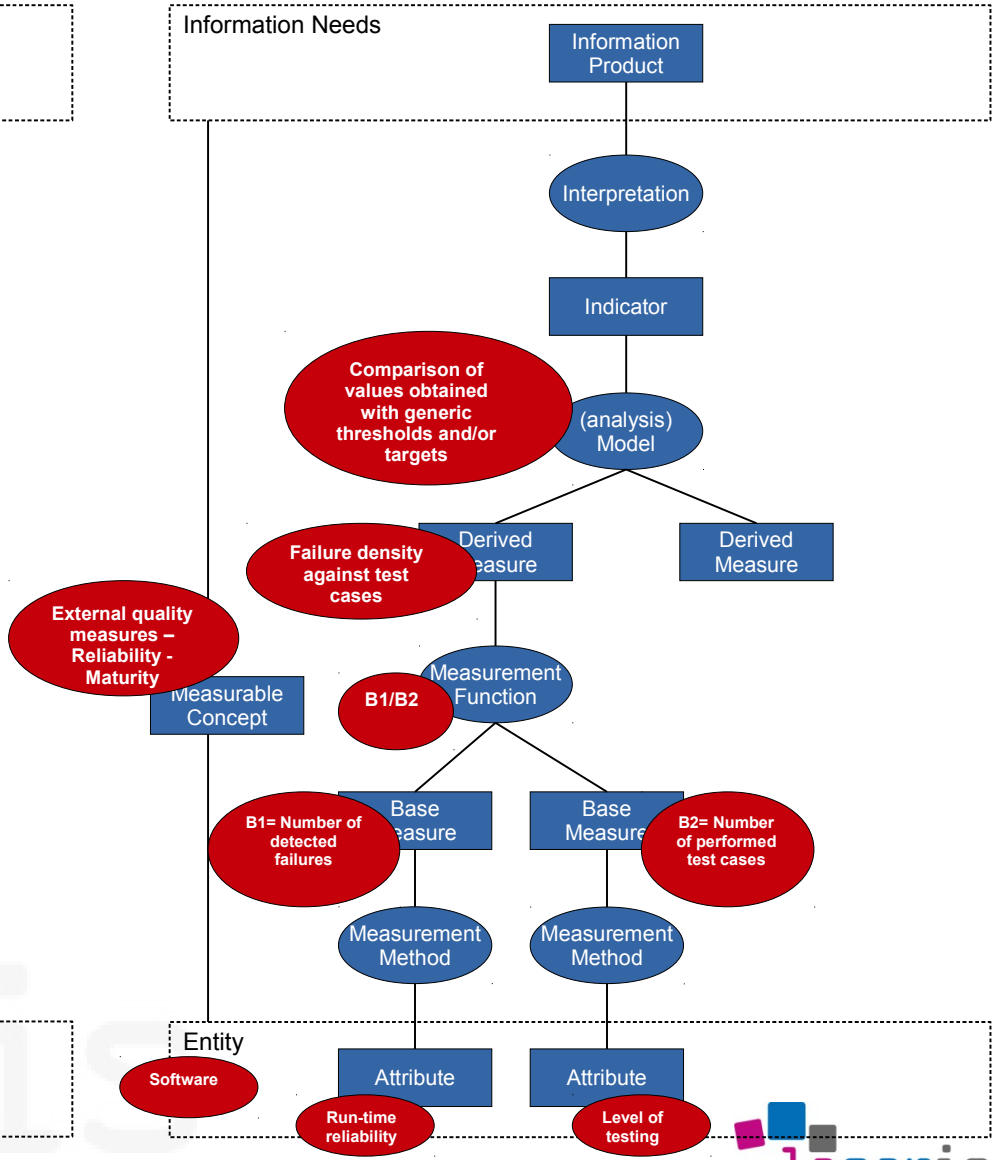
Measurement Information Model (ISO/IEC 15939)

Top part



ISO/IEC 15939 Examples





Inspired by Abran, Alain, et al. "An information model for software quality measurement with ISO standards." Proceedings of the International Conference on Software Development (SWDC-REK), Reykjavik, Iceland. 2005.

Measure Definition

- A measure is a mapping between
  - The real world
  - The mathematical or formal world with its objects and relations
- Different mappings give different views of the world depending on the context (height, weight, …)
- The **mapping relates attributes to mathematical objects; it does not relate entities to mathematical objects**

Valid Measure

- The validity of a measure **depends on definition of the attribute coherent** with the **specification of the real world**
  - Is LOC a valid measure?
  - It depends on our measurement goals, e.g.:

    → Do we consider blanks and comments in the LOCs?

    → How are the lines exactly computed (e.g. considering ";" as end statements only)

You might have two different projects with two different definitions of LOCs so that the following can be true at the same time P1>P2 and P1<P2

Valid Measures – Example (1/4)

- From Wikipedia: *"...A program with high code coverage has been more thoroughly tested and has a lower chance of containing software bugs than a program with low code coverage..."*
- Would you consider **code coverage** as a **valid measure** of **how much thoroughly one software project has been tested**?
- Suppose you have two projects and you compute code coverage

  P1 → 70% vs P2 → 80%

Would you generally consider P2 to be "better" (more accurately) tested than P1?

Valid Measures – Example (2/4)

A. Is it realistic to consider every test covering the same nr. of lines as equal?

Coverage 100%

```
[01] double div (int x, int y){
[02]    return x/y;
[03] }
```

```
AssertEquals(1.0, div(1,1));
```

Coverage 100%

```
[01] double div (int x, int y){
[02]    return x/y;
[03] }
```

```
assertEquals(0.66, div(2,3), 0.1);
```

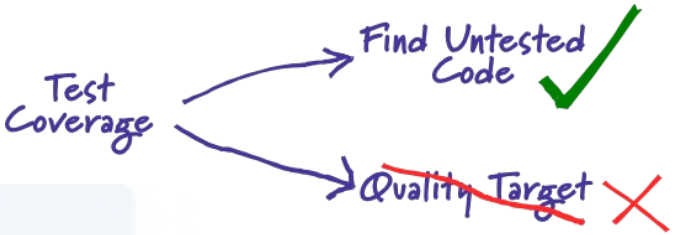
B. Is it realistic to consider every line of code as equally important for testing?

Software follows usually a Pareto principle, so that 80% of the bugs are in the 20% of the code → as well usually this code is more difficult to cover with tests

Valid Measures – Example (3/4)

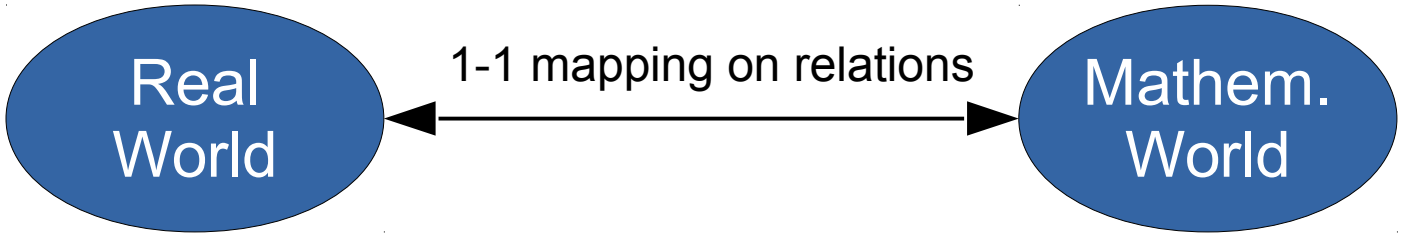
So should we “throw away” code coverage?

- According to Martin Fowler: *“Test coverage is a useful tool for finding untested parts of a codebase. Test coverage is of little use as a numeric statement of how good your tests are”*
  (http://martinfowler.com/bliki/TestCoverage.html)



Valid Measures – Example (4/4)

- What is happening in this case is that **we do not respect the representation condition**: when we assign symbols to the attributes of entities we need to preserve the meaning of relationships when moving entities from the real world to the numerical world



Measurement Scales

- The triple (A, B, μ) is called a scale
- We can have different types of scales:
  - nominal scale: $((A,\approx),(\mathbb{R}, =), \mu)$, where ≈ stands for an equivalence relation and = for a numerical relation (two objects are equivalent or not).
  - ordinal scale: $((A, \cdot\geq),(\mathbb{R}, \geq), \mu)$, where $\cdot\geq$ describes ranking properties
  - interval scale: $((A \times A, \cdot\geq),(\mathbb{R} \times \mathbb{R}, \geq), \mu)$, where $\cdot\geq$ is a preference relation about the measured objects, entities or artifacts,
  - ratio scale: $((A, \cdot\geq, \circ),(\mathbb{R}, \geq, \otimes), \mu)$, where the described axioms of an extensive structure above are valid.

Measurement Scales

- Considering the scale is quite important for the admissible operations

Scale Type	Examples	Indicators of Central Tendency
Nominal	Name of the programming language (e.g. Java, C++, C#)	Mode
Ordinal	Ranking of failures (as a measure of failure severity)	Mode + Median
Interval	Beginning date, end date of activities	Mode + Median + Arithmetic Mean
Ratio	LOC (as a measure of program size)	Mode + Median + Arithmetic Mean + geometric Mean

Morasca, Sandro. "Software measurement." Handbook of Software Engineering and Knowledge Engineering (2001): 239-276.

Measurement Scales - Examples

- Example, suppose that we have the following ranking of software tickets by severity

Level	Severity	Description
6	Blocker	Prevents function from being used, no work-around, blocking progress on multiple fronts
5	Critical	Prevents function from being used, no work-around
4	Major	Prevents function from being used, but a work-around is possible
3	Normal	A problem making a function difficult to use but no special work-around is required
2	Minor	A problem not affecting the actual function, but the behavior is not natural
1	Trivial	A problem not affecting the actual function, a typo would be an example

Measurement Scales - Examples

- Is it meaningful to use the weighted average to compare two projects in terms of severity of the open issues?

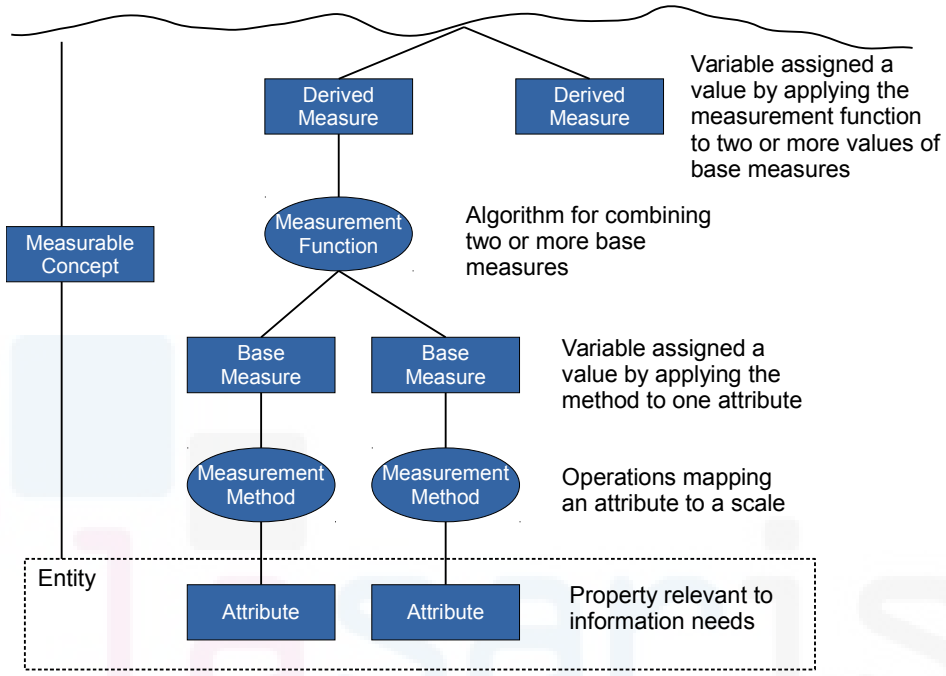
Order	Severity	P1	P2
6	Blocker	2	10
5	Critical	36	19
4	Major	25	22
3	Normal	15	32
2	Minor	2	5
1	Trivial	121	113

$$Sev(P1) = avg(2*6+36*5+25*4+15*3+2*2+121*1) = 77$$
$$Sev(P2) = avg(10*6+19*5+22*4+32*3+5*2+113*1) = 77$$

Really the projects are the same? Is there the same distance from a critical ticket to a blocker that there is between minor and trivial?

Direct vs Indirect Measures

- Some measures are harder to collect or are not regularly collected
  - **Direct:** from a direct process of measuring
  - **Indirect:** from a mathematical equation in the world of symbols



This is what in ISO/IEC 15939 we refer as base measure and derived measure

Direct vs Indirect Measures

- Direct
  - Number of known defects
- Indirect
  - Defects density (DD)

$$DD = \frac{known\ defects}{product\ \text{size}}$$

  - COCOMO, measure of effort

$$E = a \cdot KSLoC^{b} \cdot EAF$$

$$where\ b = 0.91 + 0.01 \sum_{i=1}^{5} SF_i$$

$$a = 2.94$$

Internal vs External Attributes

- It easier to collect measures of length and complexity of the code (**internal attributes of product**) than measures of its quality (**external attributes**)
  - Internal attribute: internal characteristics of product, process, and human resources
  - External attributes: due to external environment

Objective vs Subjective Measures

Objective: the same each time they are taken (e.g. automated collected by some device)

→ e.g. LOCs

Subjective: manually collected by individuals

→ e.g. time to use a functionality in an application

SOFTWARE METRICS - SIZE

lasaris

Various Measures of Size

```
[01] * multiples. Repeat until there are no more multiples
[02] * in the array.
[03] */
[04] public class PrimeGenerator
[05] {
[06]   private static boolean[] crossedOut;
[07]   private static int[] result;
[08]   public static int[] generatePrimes(int maxValue){
[09]     if (maxValue < 2){
[10]        return new int[0];
[11]     }else{
[12]        uncrossIntegersUpTo(maxValue);
[13]        crossOutMultiples();
[14]        putUncrossedIntegersIntoResult();
[15]        return result;
[16]     }
[17]   }
[18] }
```

Various Measures of Size

LOC = 18
(Lines Of Code)

CLOC=3
(Commented
Lines of Code)

```
[01] * multiples. Repeat until there are no more multiples
[02] * in the array.
[03] */
[04] public class PrimeGenerator
[05] {
[06]   private static boolean[] crossedOut;
[07]   private static int[] result;
[08]   public static int[] generatePrimes(int maxValue){
[09]     if (maxValue < 2){
[10]        return new int[0];
[11]     }else{
[12]        uncrossIntegersUpTo(maxValue);
[13]        crossOutMultiples();
[14]        putUncrossedIntegersIntoResult();
[15]        return result;
[16]     }
[17]   }
[18] }
```

Various Measures of Size

NLOC = 15 (Non-Commented Lines Of Code)

```
[01] * multiples. Repeat until there are no more multiples
[02] * in the array.
[03] */
[04] public class PrimeGenerator
[05] {
[06]   private static boolean[] crossedOut;
[07]   private static int[] result;
[08]   public static int[] generatePrimes(int maxValue){
[09]     if (maxValue < 2){
[10]        return new int[0];
[11]     }else{
[12]        uncrossIntegersUpTo(maxValue);
[13]        crossOutMultiples();
[14]        putUncrossedIntegersIntoResult();
[15]        return result;
[16]     }
[17]   }
[18] }
```

Various Measures of Size

NOC = 1
(Number Of Classes)

NOM = 1
(Number of Methods)

NOP = 1
(Number of Packages)

```
[01] * multiples. Repeat until there are no more multiples
[02] * in the array.
[03] */
[04] public class PrimeGenerator
[05] {
[06]   private static boolean[] crossedOut;
[07]   private static int[] result;
[08]   public static int[] generatePrimes(int maxValue){
[09]     if (maxValue < 2){
[10]        return new int[0];
[11]     }else{
[12]        uncrossIntegersUpTo(maxValue);
[13]        crossOutMultiples();
[14]        putUncrossedIntegersIntoResult();
[15]        return result;
[16]     }
[17]   }
[18] }
```

Measures of Size Good for?

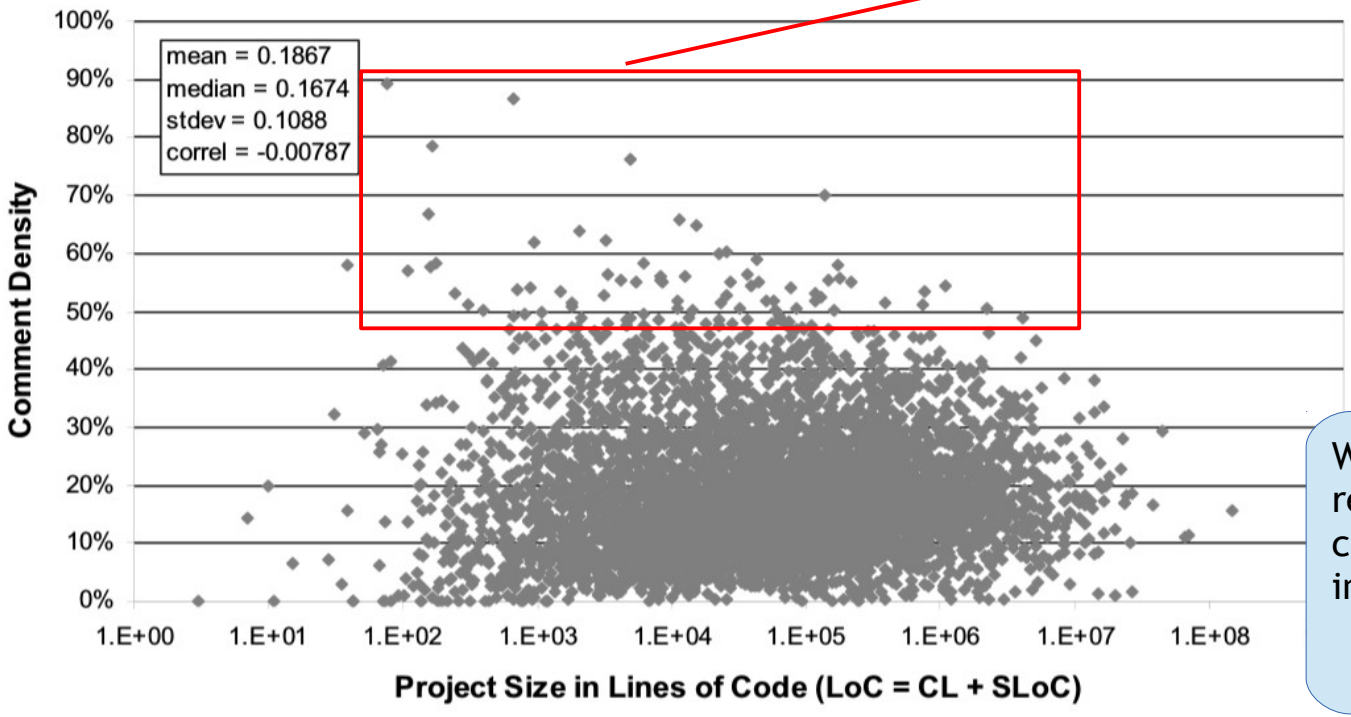
- Size is used for **normalization of existing measures**

  → from the example before, it would be much more useful to report a comments density of 16% (3/18) rather than 3 CLOCs Why?

$$CD = \frac{CLOCs}{LOCs} = \frac{3}{18} = 0.16$$

Measures of Size Good for?

- Example, using comments density to compare Open Source projects after normalization



O. Arafat and D. Riehle, "The comment density of open source software code," in 31st International Conference on Software Engineering - Companion Volume, 2009. ICSE-Companion 2009, 2009, pp. 195–198.

Measures of Size Good for?

- Size can give a good **rough initial estimation of effort**, although...

Software	LOCs
Microsoft Windows Vista	~50M
Linux Kernel 3.1	~15M
Android	~12M
Mozilla Firefox	~10M
Unreal Engine 3	~2M

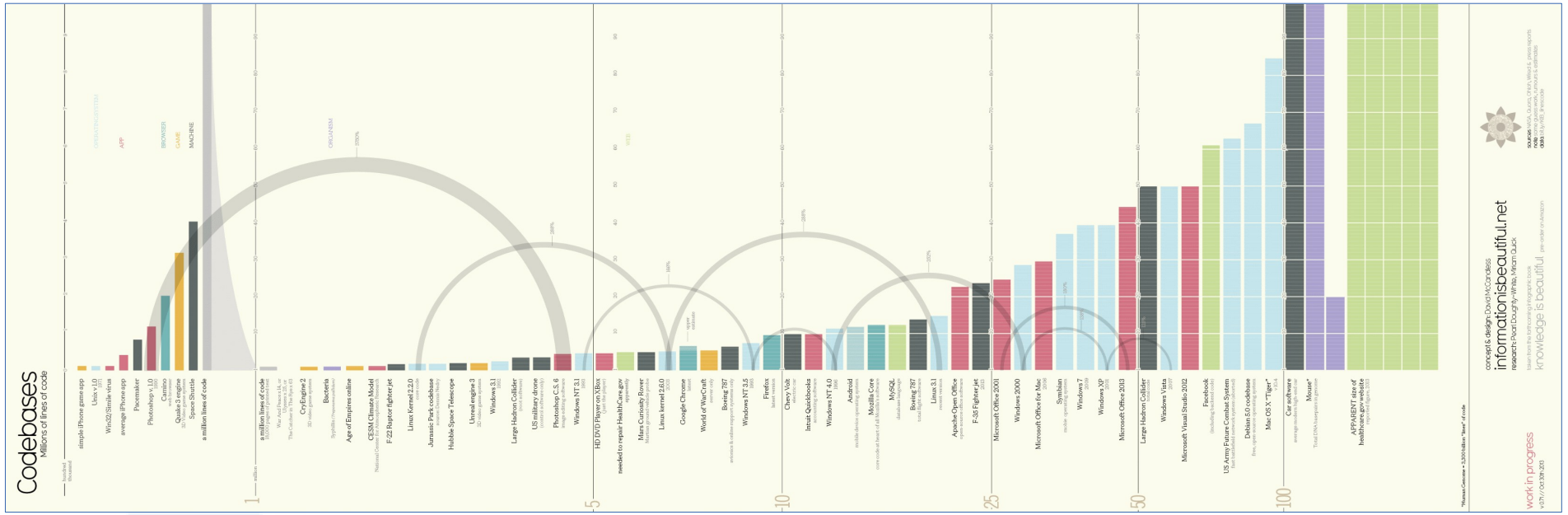
How would you compare Mozilla Firefox with the Linux Kernel in terms of maintenance effort?

- Size should NEVER be used to assess the productivity of developers

Why?

Measures of Size Good for?

- Size can be used for comparison of projects and across releases



→ http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

One Observation about LOCs

- "The task then is to refine the code base to better meet customer need. **If that is not clear, the programmers should not write a line of code. Every line of code costs money to write and more money to support.**"

Jeff Sutherland, one of the main proponents of the Agile Manifesto and the SCRUM methodology

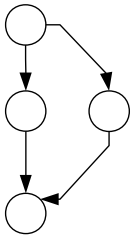
SOFTWARE METRICS - COMPLEXITY

lasaris

Cyclomatic Complexity (CC)

CC = 3

Number of decision points - if, while, for, foreach, case, default, continue, goto, &&, ||, catch, ? : (ternary operator), ??(nonnull operator)



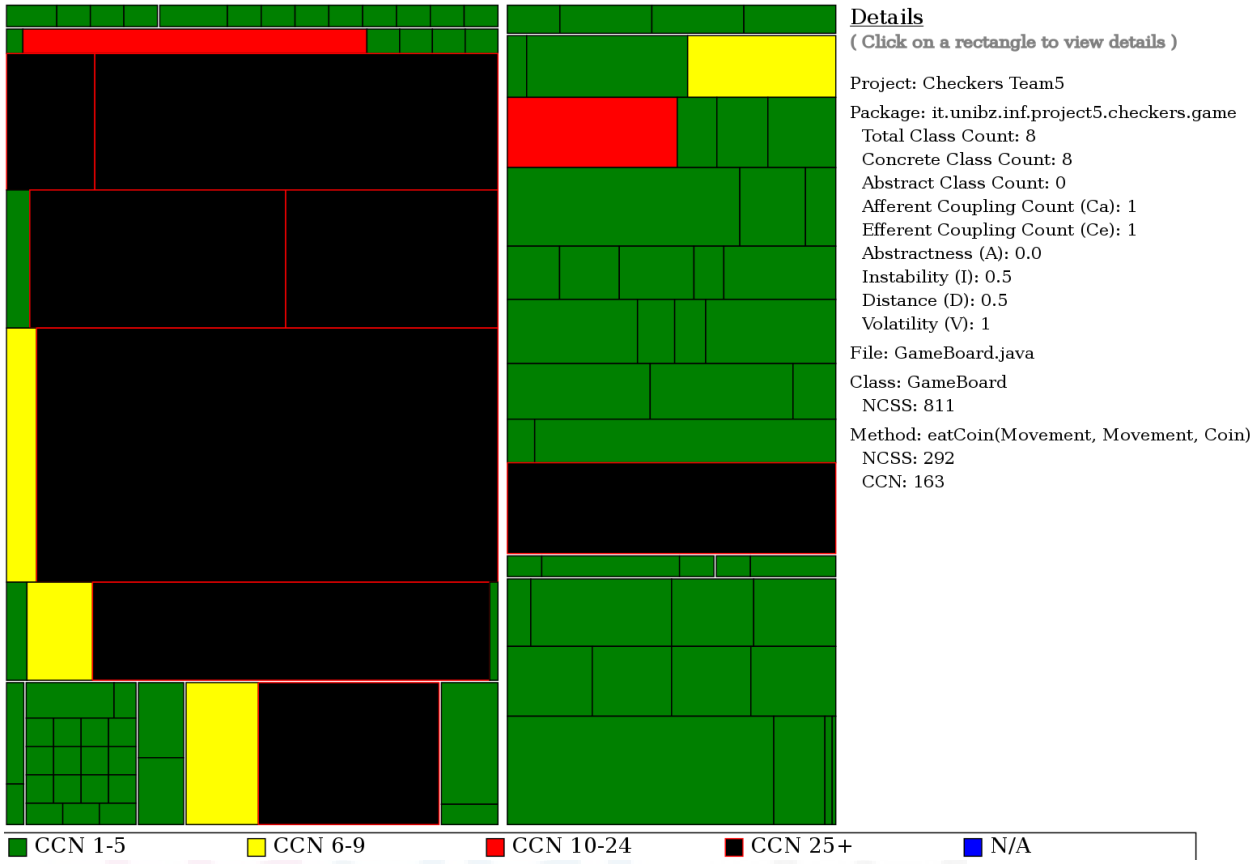
v(G) = e – n + 2

Typical ranges
1-4 low
5-7 medium
8-10 high
11+ very high

```
[01] * multiples. Repeat until there are no more multiples
[02] * in the array.
[03] */
[04] public class PrimeGenerator
[05] {
[06]   private static boolean[] crossedOut;
[07]   private static int[] result;
[08]   public static int[] generatePrimes(int maxValue){
[09]     if (maxValue < 2){
[10]       return new int[0];
[11]     }else{
[12]       uncrossIntegersUpTo(maxValue);
[13]       crossOutMultiples();
[14]       putUncrossedIntegersIntoResult();
[15]       return result;
[16]     }
[17]   }
[18] }
```

Example by using CC

- The following code structure from a 2008 students' project implementing chess: one method with 292LOCs and 163 CC



Example by using CC

- Let's decompose a bit such huge method

```
public boolean eatCoin(Movement mov, Movement eatMov, Coin coin)
throws IOException{
        //Controls if the eatMove is in the board, if not return
        if(!canMove(eatMov)){
            System.out.println("You can't eat this coin");
            return false;
        }

        try{
            //If it is a coin
            if(!this.board[mov.row][mov.col].isKing()){
                //If the coin to eat isn't a king
                System.out.println("nextRow " + mov.nextRow + "
                     nextCol " + mov.nextCol + " isKing " +
                     this.board[mov.nextRow][mov.nextCol].isKing());
                if(!this.board[mov.nextRow][mov.nextCol].isKing()){
                     ....
```

Example by using CC

```
        //White king
        if(coin.checkColour() == -1){
            //If more then one coin can be eat the plaer have to make a choose
            if(((checkField(tempMov1) == 1 && checkField(newEatMov1) == 0) || (checkField(tempMov2) == 1 && checkField(newEatMov2) == 0)) && ((checkField(
            == 1 && checkField(newEatMov3) == 0) || (checkField(tempMov4) == 1 && checkField(newEatMov4) == 0)) && ((checkField(tempMov1) == 1 &&
            checkField(newEatMov1) == 0) || (checkField(tempMov3) == 1 && checkField(newEatMov3) == 0)) && ((checkField(tempMov2) == 1 && checkField(newEa
            0) || (checkField(tempMov4) == 1 && checkField(newEatMov4) == 0))){
                window.moveCoin(window.nextYClick, window.nextXClick);
                window.preXClick = window.nextXClick;
                window.preYClick = window.nextYClick;
                window.secondClick = false;
                window.anzClick = 1;
                window.jTextArea.setText("Scegli che pedina mangiare");
                while(!window.secondClick){}
                if((window.nextXClick/50==tempMov1.nextCol && window.nextYClick/50==tempMov1.nextRow) || (window.nextXClick/50==newEatMov1.nextCol &&
                window.nextYClick/50==newEatMov1.nextRow)){
                    eatCoin(tempMov1, newEatMov1, coin);
                }
                else{
                    if((window.nextXClick/50==tempMov2.nextCol && window.nextYClick/50==tempMov2.nextRow) || (window.nextXClick/50==newEatMov2.nex
                    window.nextYClick/50==newEatMov2.nextRow)){
                        eatCoin(tempMov2, newEatMov2, coin);
                    }
                    else{
                        if((window.nextXClick/50==tempMov3.nextCol && window.nextYClick/50==tempMov3.nextRow) ||
                        (window.nextXClick/50==newEatMov3.nextCol && window.nextYClick/50==newEatMov3.nextRow)){
                            eatCoin(tempMov3, newEatMov3, coin);
                        }
                        else{
                            if((window.nextXClick/50==tempMov4.nextCol && window.nextYClick/50==tempMov4.nextRow) ||
                            (window.nextXClick/50==newEatMov4.nextCol && window.nextYClick/50==newEatMov4.nextRow)){
                                eatCoin(tempMov4, newEatMov4, coin);
                            }
                            else{
                                boolean ret = false;
                                while(!ret){
                                    i = (int) (Math.random() * 4);
                                    switch(i){
                                    case 1:
                                        if(checkField(tempMov1) == 1 && checkField(newEatMov1) == 0){
                                            window.nextXClick = tempMov1.nextCol;
                                            window.nextYClick = tempMov1.nextRow;
                                            eatCoin(tempMov1, newEatMov1, coin);
                                            ret = true;
```

Complexity

- A word of warning is that metrics take typically into account syntactic complexity **NOT semantical complexity**
- Both of the following code fragments have the same Cyclomatic Complexity

```
[04] public class PrimeGenerator
[05] {
[06]   private static boolean[] crossedOut;
[07]   private static int[] result;
[08]
[09]   public static int[] generatePrimes(int maxValue){
[10]     if (maxValue < 2){
[11]        return new int[0];
[12]     }else{
[13]        uncrossIntegersUpTo(maxValue);
[14]        crossOutMultiples();
[15]        putUncrossedIntegersIntoResult();
[16]        return result;
[17]   }
[18] }
```

```
[04] public class A
[05] {
[06]   private static boolean[] c;
[07]   private static int[] b;
[08]
[09]   public static int[] generate(int m){
[10]     if (m < 2){
[11]        return new int[0];
[12]     }else{
[13]        methodOne(m);
[14]        methodTwo();
[15]        methodThree();
[16]        return b;
[17]   }
[18] }
```

- As well, **as in the initial motivating example**, a word of warning when **comparing projects in terms of average complexity**

OBJECT ORIENTED METRICS

lasaris

ndepend metrics

Version 1.1

References

www.ndepend.com | Documentation | Metrics definitions

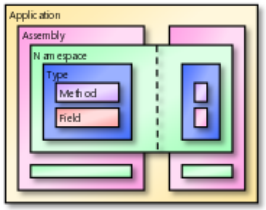
Agile Principles, Patterns, and Practices in C#, Robert C. Martin, Prentice Hall PTR, 2006

metrics

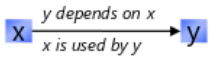
	Application	Assembly	Namespace	Type	Method	Field
cardinality						
Lines of Code (LOC) [1]	●	●	●	●	●	
Lines of Comments [2,3]	●	●	●	●	●	
Percentage Comment [2]	●	●	●	●	●	
Number of IL Instructions [4]	●	●	●	●	●	
Number of Assemblies	●					
Number of Namespaces [5]	●	●				
Number of Types	●	●	●			
Number of Fields	●	●	●	●		
Number of Methods	●	●	●	●		
Number of Parameters					●	
Number of Variables					●	
relational						
Afferent Coupling (Ca)		●	●	●	●	●
Efferent Coupling (Ce)		●	●	●	●	●
Instability (I)		●				
Relational Cohesion (H)		●				
Abstractness (A)		●				
Distance from main sequence (D)		●				
Level		●	●	●		
Rank		●	●	●		
Cyclomatic Complexity (CC)				●	●	
IL Cyclomatic Complexity (ILCC)				●	●	
Lack of Cohesion of Methods (LCOM)				●		
Instance Size				●		●
Association Between Classes (ABC)				●		
Number of Children (NOC)				●		
Depth of Inheritance Tree (DIT)				●		
Response for a Type (RFT)				●		

[1] *Requires PDBs. Logical LOC: number of IL sequence points; language and style independent.*
[2] *Require source code.*
[3] *Currently for C# only, VB soon. Metric is not additive.*
[4] *Varies depending on compiling for release or debug.*
[5] *One namespace defined over N assemblies counts as N namespaces*

packages



key



coupling

Efferent coupling (Ce): number of types within this package that depend on types outside this package

Afferent coupling (Ca): number of types outside this package that depend on types within this package

instability

Instability (I): ratio of efferent coupling to total coupling, which indicates the package's resilience to change.

I = Ce / (Ce + Ca)

I=0 indicates a completely stable package, painful to modify.
I=1 indicates a completely instable package.

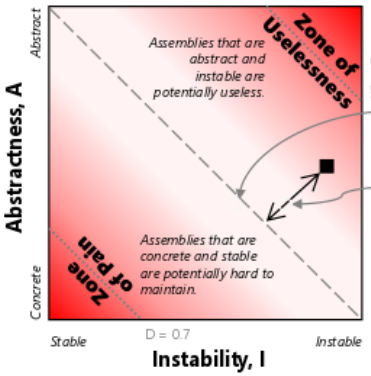
abstractness

Abstractness (A): ratio of the number of internal abstract types to the number of internal types.

A=0 indicates a completely concrete package.

A=1 indicates a completely abstract package .

distance from main sequence: zone of pain and zone of uselessness



Main sequence, **A + I = 1**, represents optimal balance between abstractness and stability

D is the normalized distance from main sequence, **0 ≤ D ≤ 1**

Assemblies where D > 0.7 might be problematic. However, in the real world it is very hard to avoid such assemblies. Allow a small percentage of your assemblies to violate this constraint.

cohesion

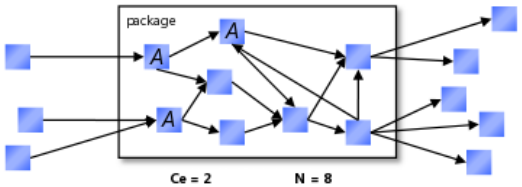
Relational Cohesion (H): average number of internal relationships per type:

H = (R + 1) / N, where

R = number of type relationships internal to the package, and

N = number of types in the package.

Classes inside an assembly should be strongly related, the cohesion should be high. On the other hand, too high values may indicate over-coupling. A good range is 1.5 ≤ H ≤ 4.0.



Ce = 2
Ca = 5
I = 2/7 = 0.29

N = 8
R = 12
A = 3/8 = 0.375
H = 13/8 = 1.625

depth of inheritance tree

The depth of inheritance tree (**DIT**) for a class or a structure is its number of base classes (including *System.Object* thus DIT ≥ 1).

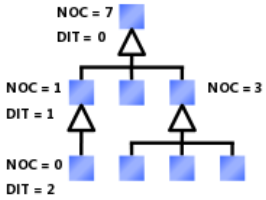
Types where DIT > 6 might be hard to maintain.

Not a rule since sometime classes inherit from tier classes which have a high DIT. E.g., the average depth of inheritance for framework classes which derive from System.Windows.Forms.Control is 5.3.

number of children

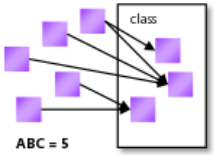
Number of children (**NOC**) for a class is the number of types that subclass it directly or indirectly.

Number of children for an interface is the number of types that implement it.



association between classes

The association between classes (**ABC**) is the number of members of others types that a class directly uses in its the body of its methods.



rank

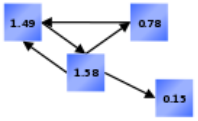
Google Page Rank applied to types or methods.

If $T_1, ..., T_N$ are the types (methods) that depend on type (method) A, then the rank of A is

$$R(A) = (1-d) + d\sum_{i=1}^{N} \frac{R(T_i)}{Ca(T_i)}$$

d = damping factor, typically 0.85.

Test types with high rank thoroughly, as defects there are likely to be more catastrophic.

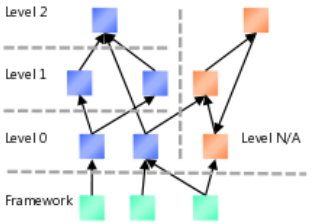


level

If a package depends on nothing or framework packages, then it is Level 0.

If a package depends on packages of at most Level N, then it is Level N+1.

If a package is part of a circular dependency, then it is Level N/A. If a package depends on something of Level N/A, it is Level N/A.



lack of cohesion of methods

The single responsibility principle states that a class should not have more than one reason to change. Such a class is cohesive.

$$LOCM = 1 - \frac{\sum_{f \in F} |M_f|}{|M| \times |F|}$$

M = static and instance methods in the class,

F = instance fields in the class,

M_f = methods accessing field *f*, and

|*S*| = cardinality of set *S*.

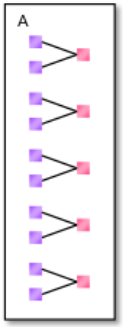
In a class that is utterly cohesive, every method accesses every instance field

$$\sum |M_f| = |M| \times |F|$$

so LOCM = 0.

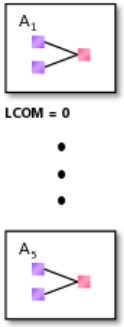
A high LCOM value generally pinpoints a poorly cohesive class.

Types where LCOM > 0.8 and |F| > 10 and |M| >10 might be problematic. However, it is very hard to avoid such non-cohesive types.



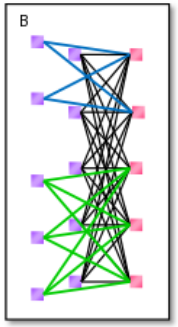
LCOM = 0.8

One class with five fields, each with a getter and setter.



LCOM = 0

Five classes, each with one field and a getter and setter.



LCOM = 0.24

Five constructors each set five fields (black); two getters that access two fields (blue); and three getters that access three fields (green).

cyclomatic complexity

The number of decisions that can be taken in a procedure.

Cyclomatic Complexity (CC)

Number of these expressions in the method body:

if, while, for, foreach, case, default, continue, goto, &&, ||, catch, ? : *(temary operator)*, ?? *(nonnull operator)*

These expressions are not counted:

else, do, switch, try, using, throw, finally, return, *object creation, method call, field access*

CC > 15 are hard to understand, CC > 30 are extremely complex and should be split into smaller methods (unless generated code)

IL Cyclomatic Complexity (ILCC)

Number of distinct code offsets targeted by jump/branch IL instructions. Language independent.

ILCC is generally larger than CC.

ILCC(**if**) = 1

ILCC(**for**) = 2

ILCC(**foreach**) = 3

ILCC > 20 are hard to understand, ILCC > 40 are extremely complex and should be split into smaller methods (unless generated code)

The Goal Question Metrics (GQM) Approach

Software Measurement - Pitfalls

- Common pitfalls in software measurement
  - Collecting measurements without a meaning
    - Measurement must be goal-driven
  - Not analyzing measurements
    - Numbers need detailed analysis
  - Setting unrealistic targets
    - Targets should not be uniquely defined based on the numbers
  - Paralysis by analysis
    - Measurement is a key activity in management, not a separate activity

Count what is countable.
Measure what is measurable.
And what is not measurable, make measurable.
Galileo Galilei

The GQM Approach

- Introduced in 1986 by Rombach and Basili
  - GQM stands for Goal Question Metric
- It is a deductive instrument to derive suitable measures from prescribed goals
- The paradigm is initiated by Business Goals (BG)

Examples of Business Goals

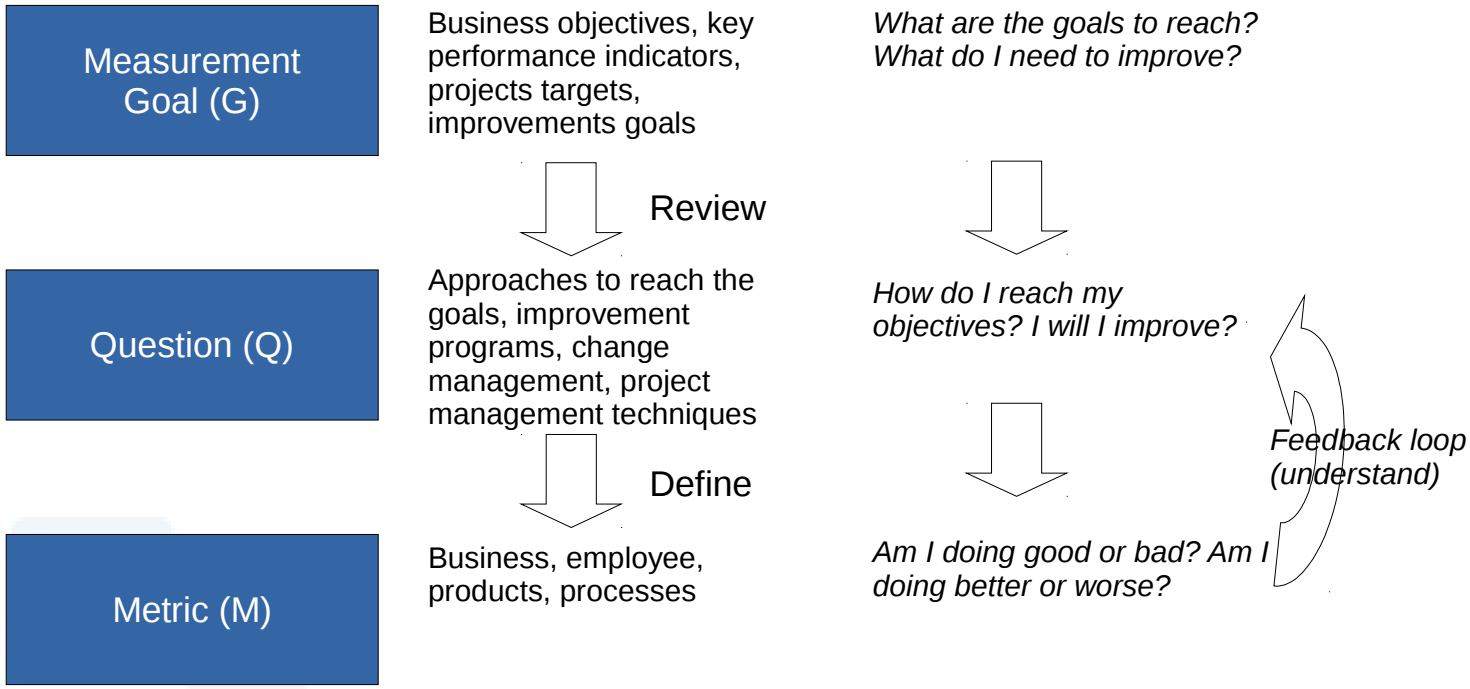
- Improve the quality of a software product
- Understand the development process for a given project
- Enhance the inspection process in the testing phase
- Decide on the adoption of a new software tool
- Evaluate costs of a transition to a new sw solution
- Assess the efficiency of the development process
- Evaluate the current testing strategy

The GQM Approach

- From the BGs we can derive the GQM
- The Goal Question Metric top-down approach consists of three layers
  - **Conceptual layer** – the Measurement Goal (G)
  - **Operational layer** – the Question (Q)
  - **Measurement layer** – the Metric (M)

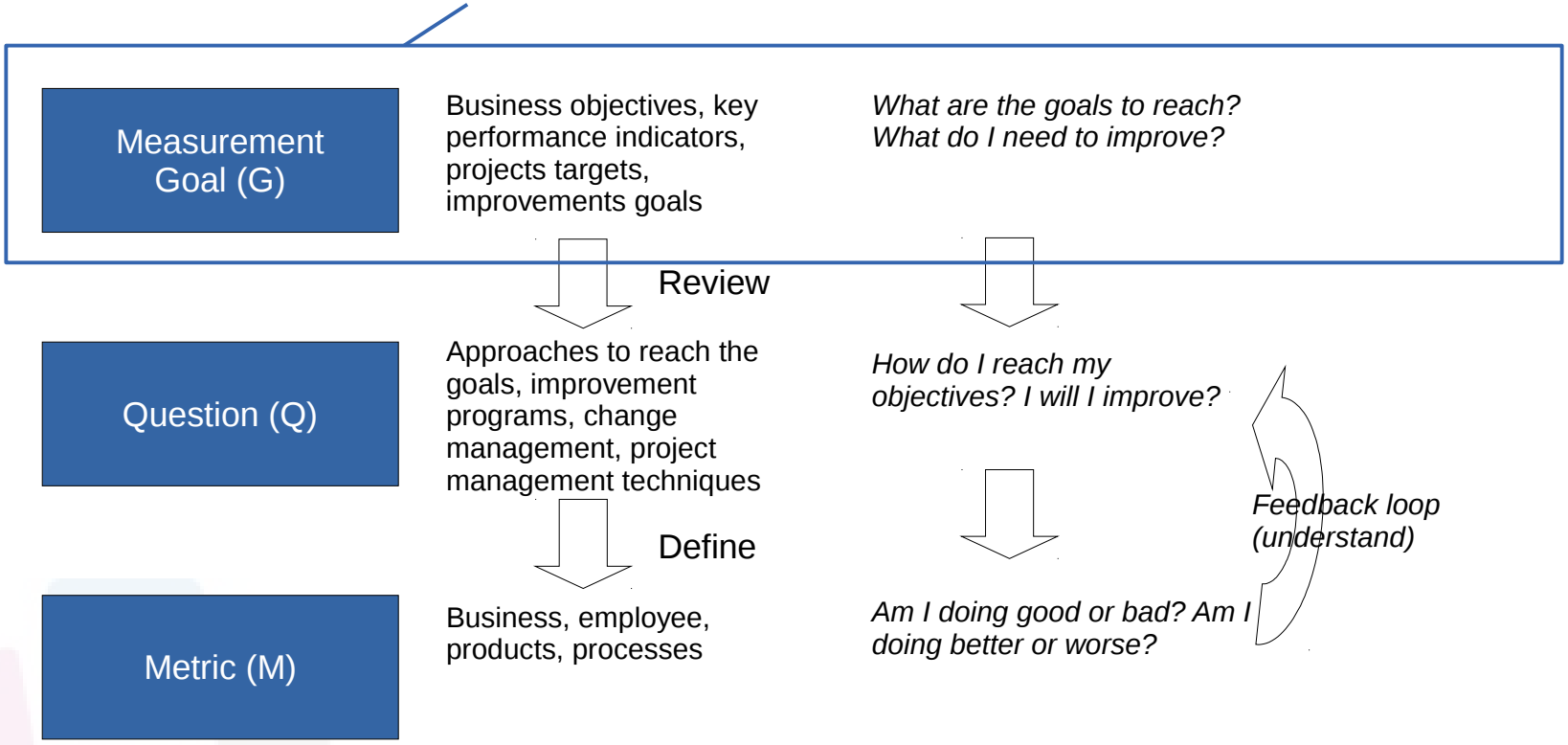
Goal-oriented Measurement

- Measurements must be goal-oriented
- Following typically a structure as the GQM approach:



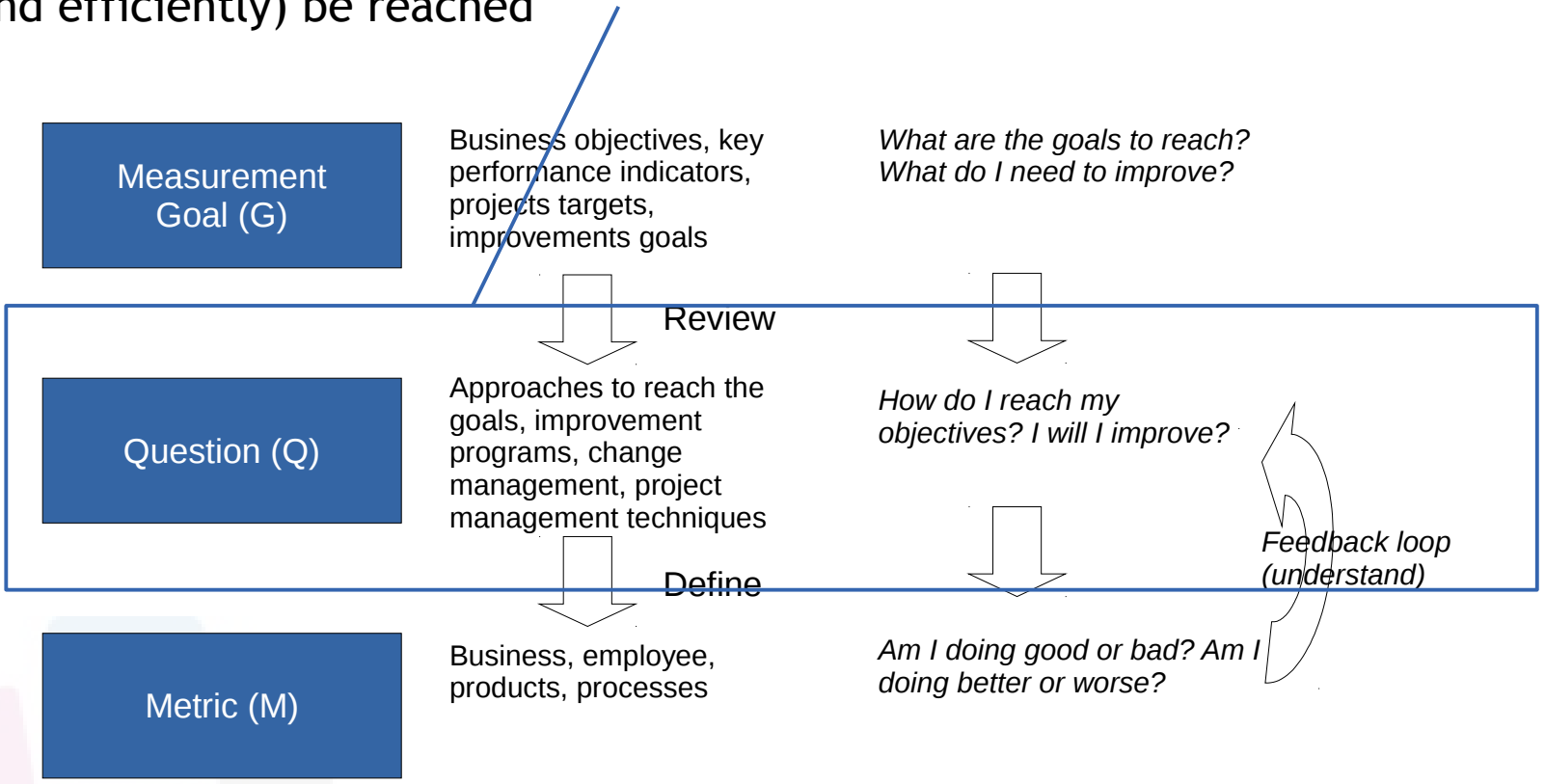
Goal-oriented Measurement

Starting with objectives which can be personal or company-wide it is determined what to improve. Goals are translated into what should be achieved in the context of a software project or process or product



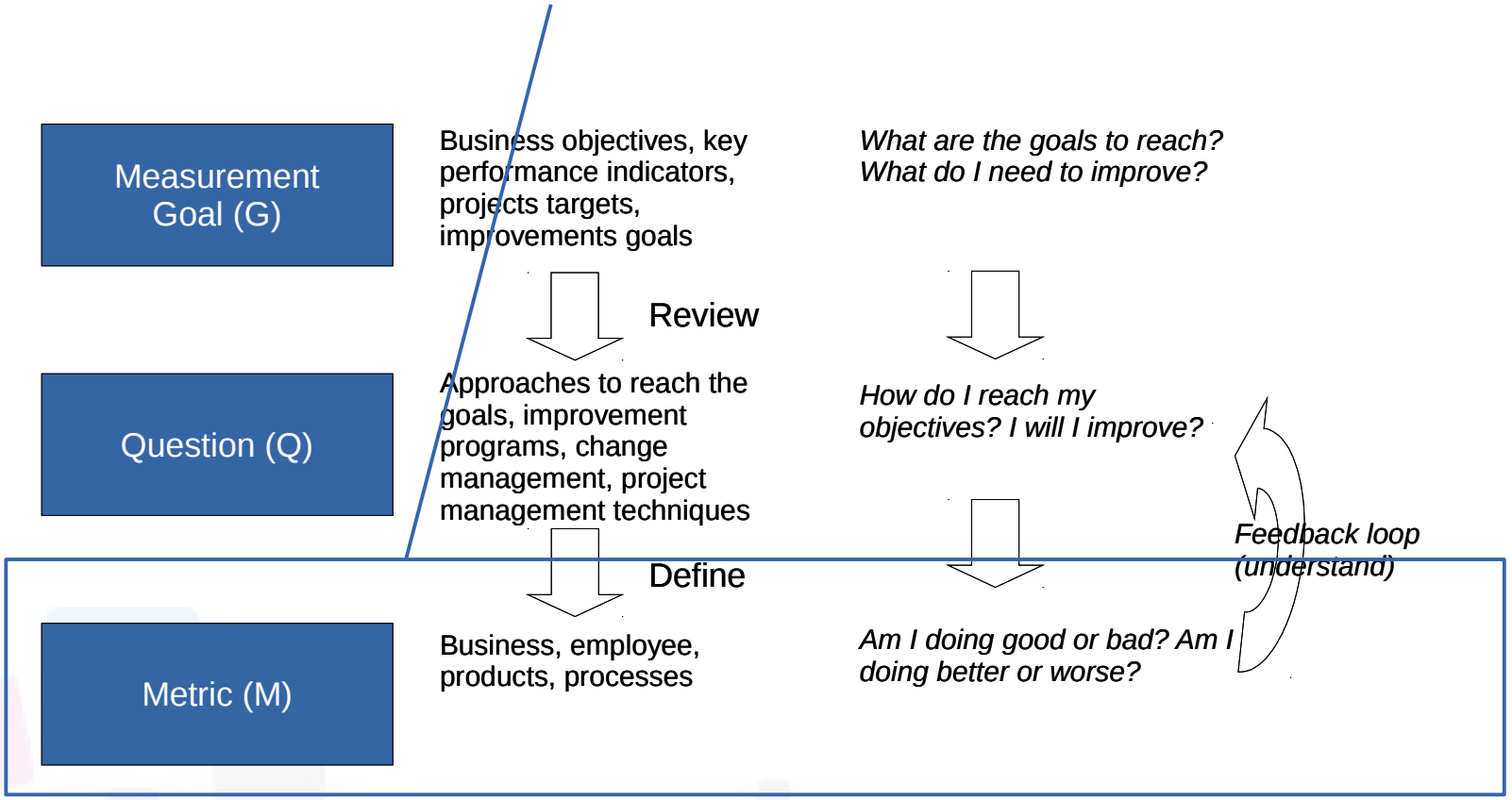
Goal-oriented Measurement

Identification about how the improvement should be done
Asking questions helps in clarifying how the objectives of step 1 will effectively (and efficiently) be reached



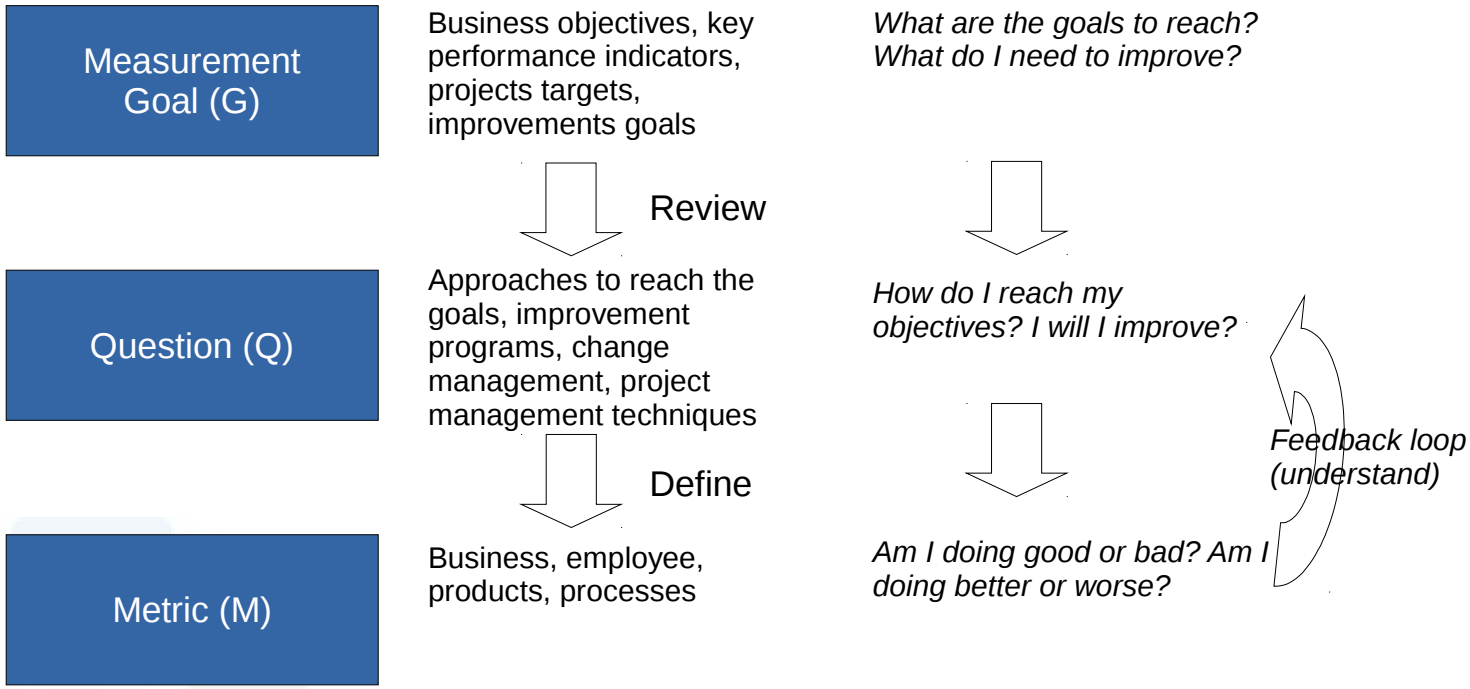
Goal-oriented Measurement

Identify appropriate measurements that will indicate progress and whether the change is pointing in a good direction



Goal-oriented Measurement

The primary question must be **“What do I need to improve?”** rather than “What measurements should I use?”
Software measurements should follow from the organizational needs



The Measurement Goal

- The MG is structured in 5 items
  - **Object of Study (OS)**: what we want to measure - as a model
  - **Purpose**: is the major verb
  - **Focus (F)**: the perspective to which one looks at the OS
  - **Point of view**: generally is a person or a category of people
  - **Context**: the environment in which the OS is observed

The Measurement Goal

- Here are some possible and common used words for each item of the Goal structure
- **Object of study**: process, product, model, metric, etc
- **Purpose**: characterize, evaluate, predict, motivate, etc. in order to understand, assess, manage, engineer, improve, etc. it
- **Point of view**: manager, developer, tester, customer, etc.
- **Perspective or Focus**: cost, effectiveness, correctness, defects, changes, product measures, etc.
- **Environment or Context**: specify the environmental factors, including process factors, people factors, problem factors, methods, tools, constraints, etc.

The Questions - Example

- The Question is a link between OS and F

- BG_1: *improve the software inspection process*
- MG_1: *Analyze the current inspection process to evaluate it in terms of duration testing from the point of view of the testers in a small software house*
  - OS: **Inspection method**
  - Focus: **cost**
  - Q Link: *weekly labor of a tester to inspect a code*
- Q_1: *What is the cost of the weekly labor of a tester to inspect a code with the given process?*

The Metrics - Example

- Metrics are a set of measures for OS, F, and the QL

- Example
- I can derive the following metrics

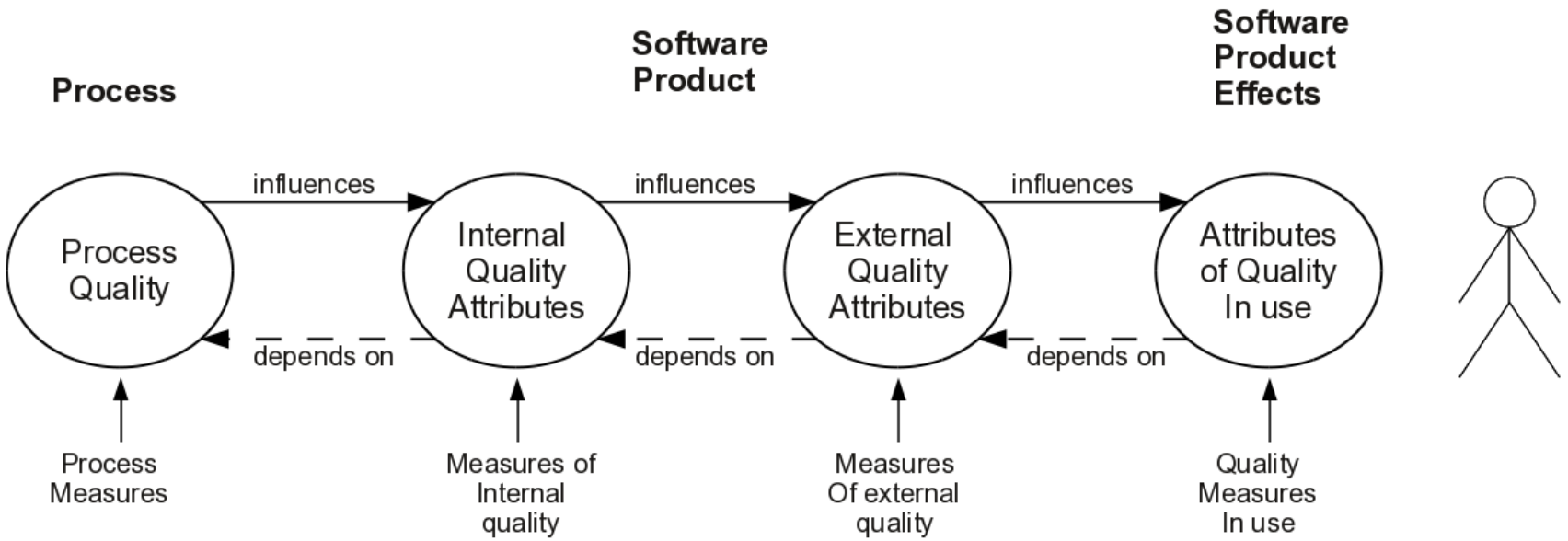
M1= weekly salary * effort * # testers

M2= weekly salary * effort * duration of the inspection

Software Measurement & the role in Software Quality Improvement

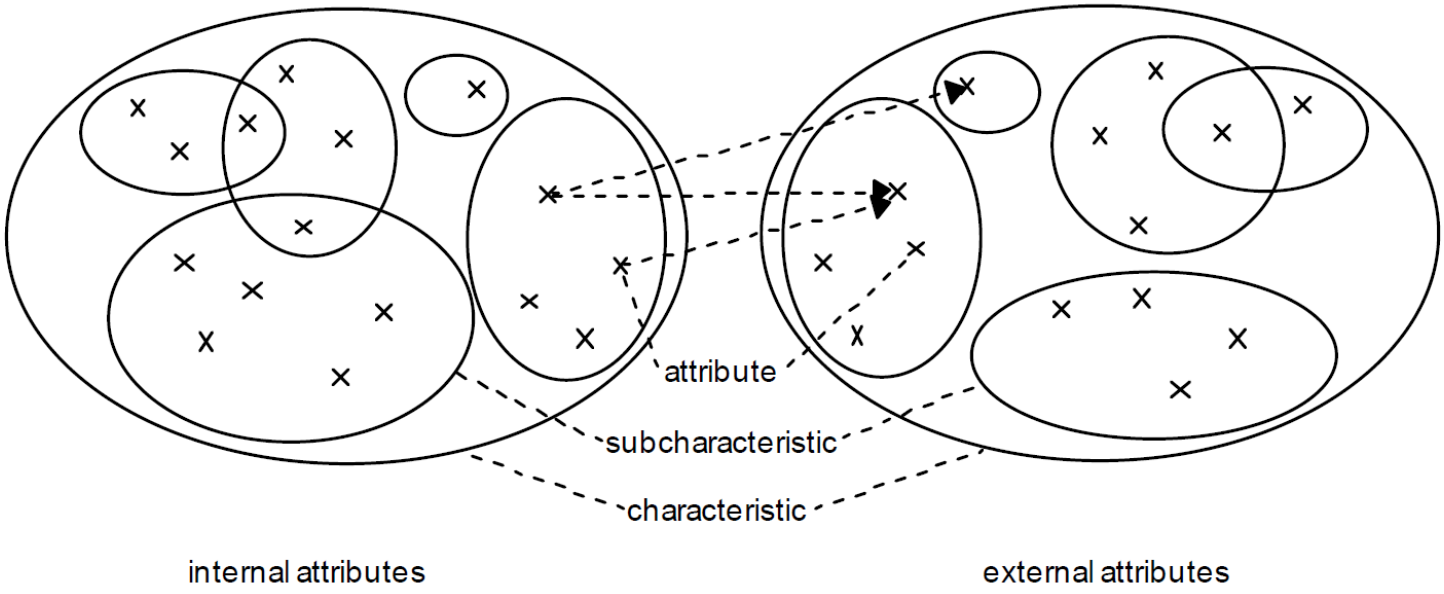
External Product Measures

- One of the aims of Software Engineering is to improve the quality of the software



External Product Measures

- The mapping of internal attributes to external ones – and then quality in use – is not as straightforward

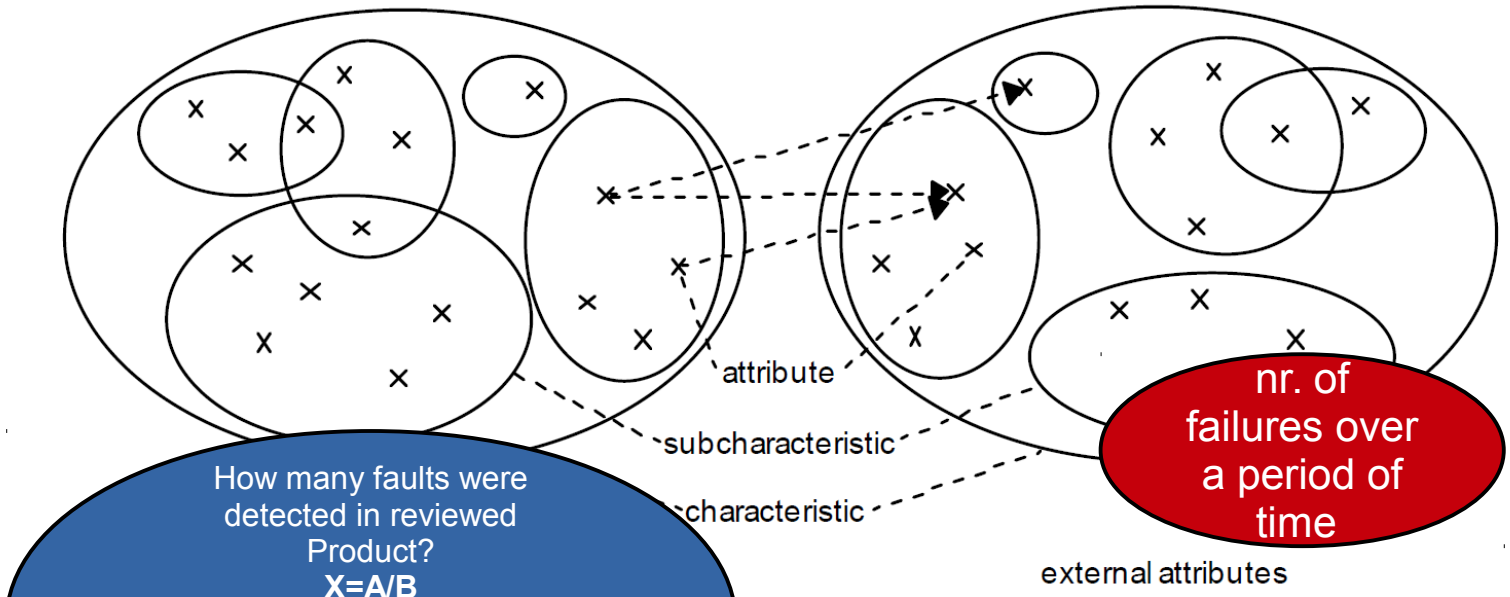


Example - Reliability

- Concept that dates back to hardware reliability
  - But software has a different behavior
  - Ideas never wear out they do not deteriorate as they are not bounded to a physical object
- A system is said reliable if it operates in an external environment following the prescribed specifications
- A **failure** is a deviation from the prescribed flow
- The concept is depending on time: a system is reliable in a given interval of time
  - Reliability is traditionally measures by the number of occurrences of failures (in time)
  - There exists no software product with zero defects

Example - Reliability

- The mapping of internal attributes to external ones – and then quality in use – is not as straightforward



Internal Measures of Reliability

- Failures are difficult to trace
- They depend on the environment
- They depend by the end-users
- Failures are hardly collected
  - Automatic or autonomous collection
    - They may contain useless information
    - Big effort to clean the data
- Use of internal causes of failures
  - Defects, bugs, faults, errors
- Hope: **fix internal mistakes to fix the corresponding failure(s)**

Problems

- Intervening to fix a bug may inject new bugs (hence failures) in the code
  - The same happens in the design, architecture, test
- Testing the code to find failures cannot reproduce all the users' behaviour (in vitro testing)
- Inspecting the code is expensive
- It is not proved that there is a clear cause effect relation between defects and failures
  - A failure is caused by defects
  - A defect might not cause a failure in the time period in which the application is used
  - Pareto principle: The 20% of the classes are responsible of the 80% of the failures

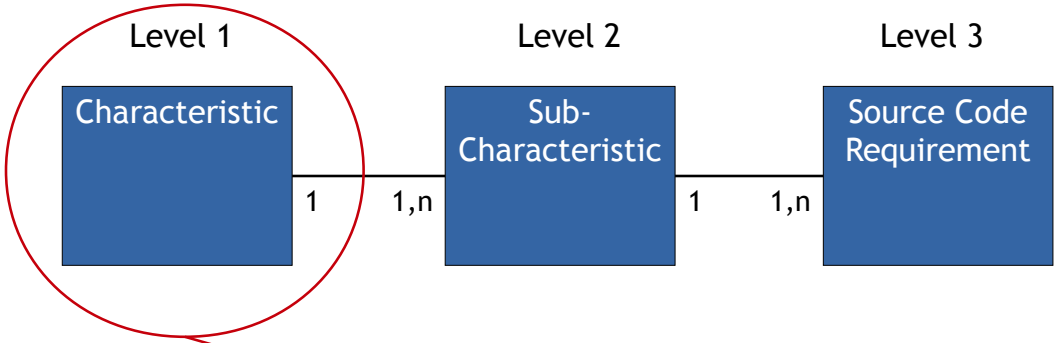
SQALE (Software Quality Assessment Based on Lifecycle Expectations)

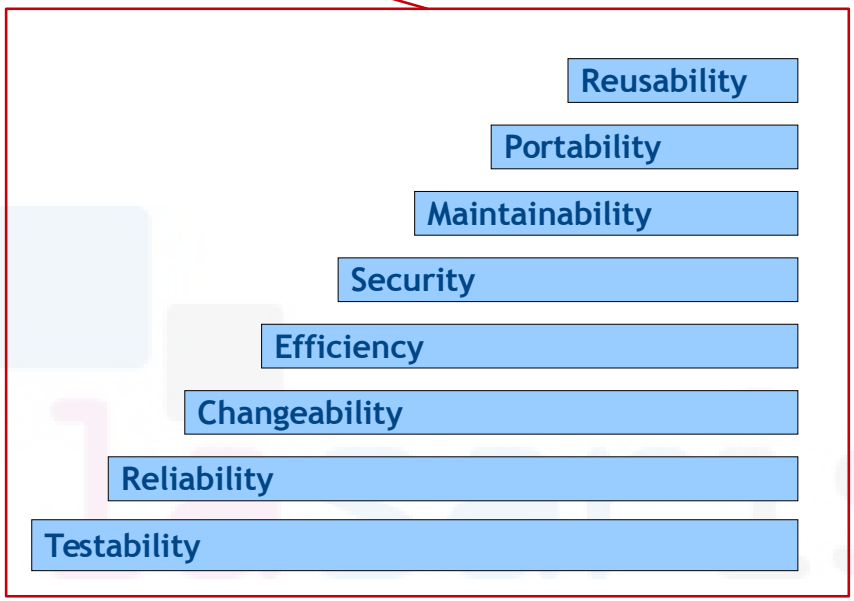
SQALE

- SQALE (Software Quality Assessment Based on Lifecycle Expectations) is a quality method to **evaluate technical debts** in software projects **based on the measurement of software characteristics**
- It allows to discuss here how quality characteristics have been mapped into numerical representations

SQALE

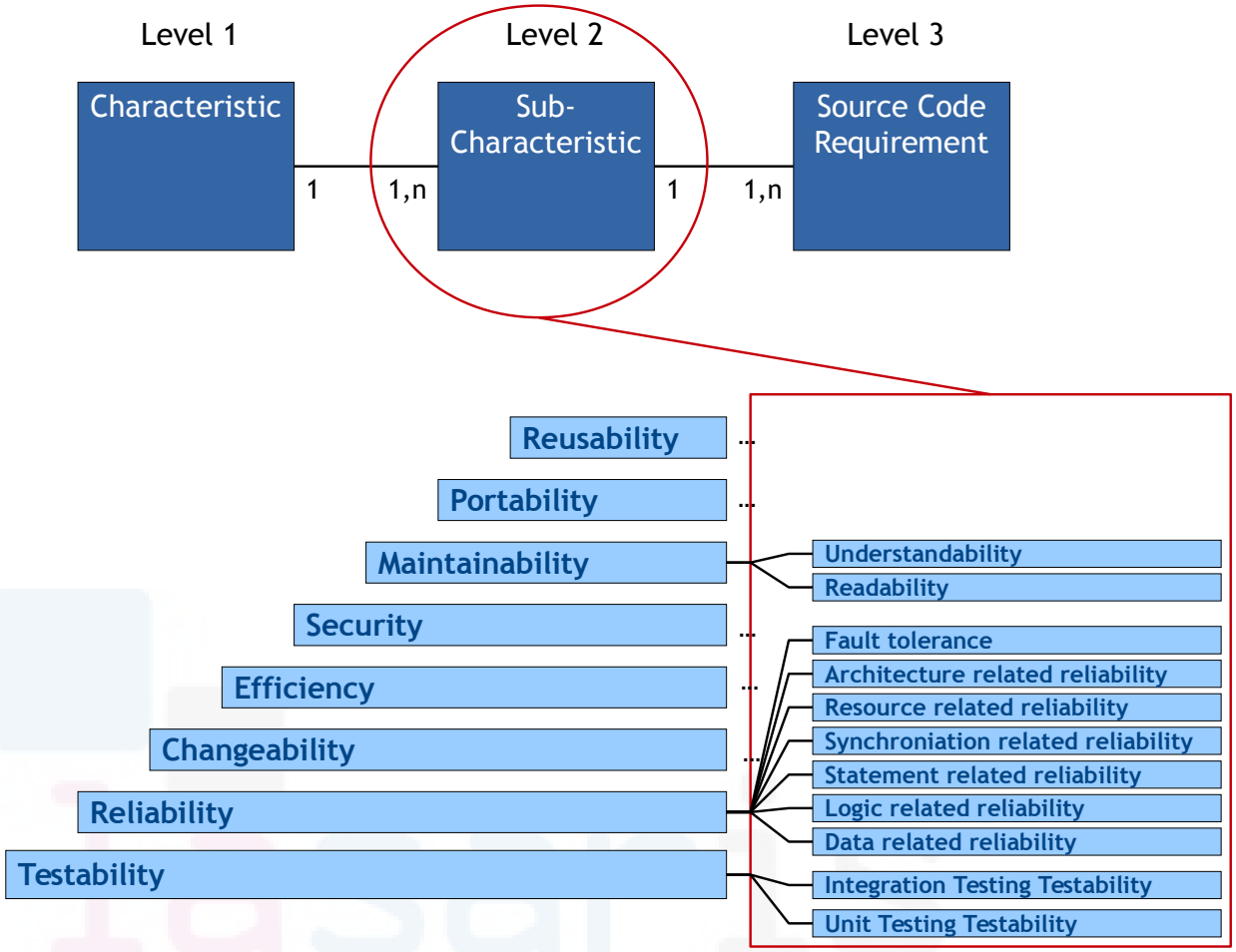
- SQALE quality model is based around three levels, the first one including 8 software characteristics





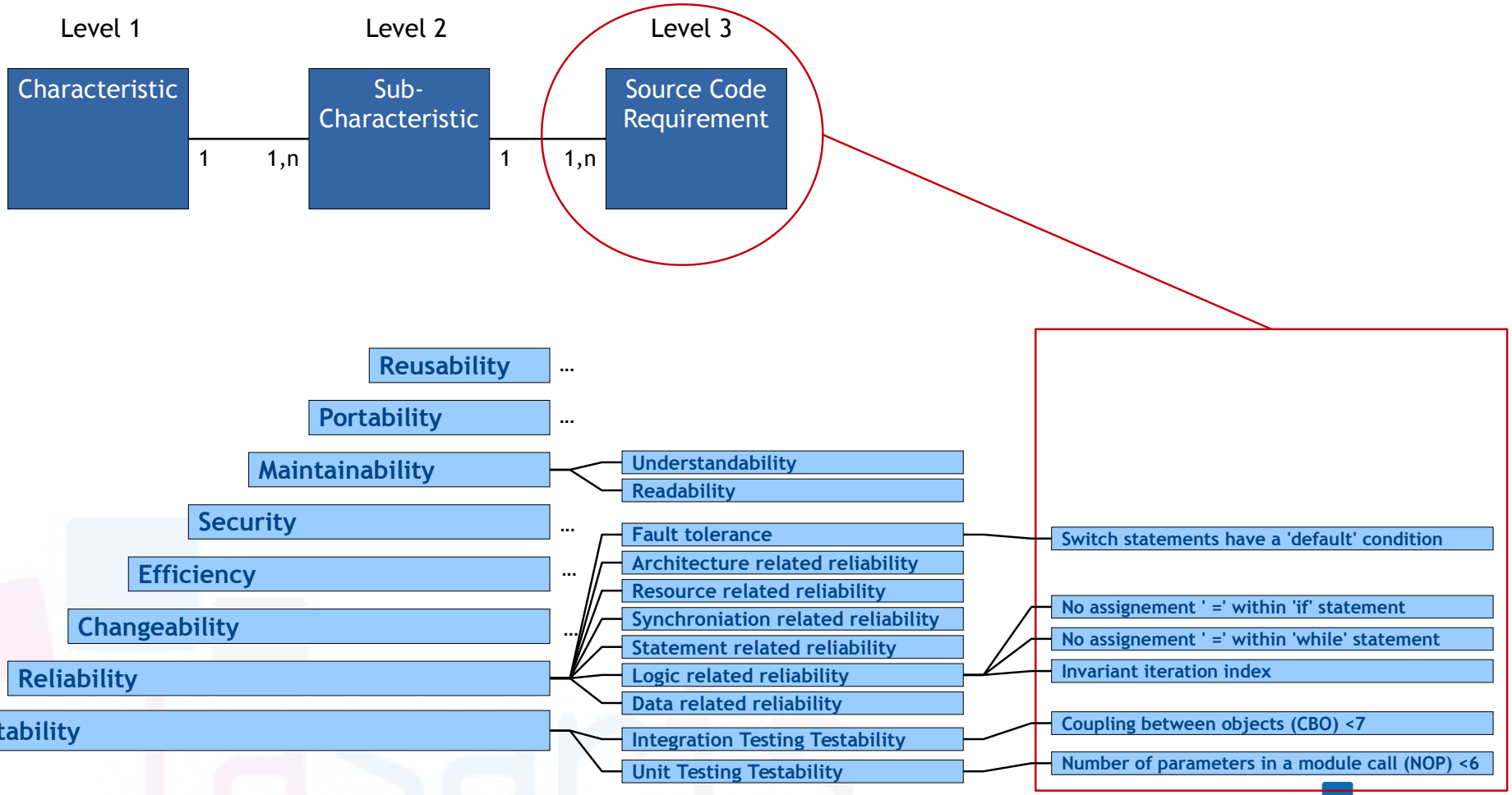
SQALE

- The second level is formed by characteristics



SQALE

- The third level is linking language specific constructs to the sub-characteristics



SQALE – Remediation Function

- For each of the source code requirements we need to associate a **remediation function** that translates the non-compliances into remediation costs
- In the most complex case you can associate a different function for each requirement, but in the most simple case you can have some predefined value for categories in which code requirements are in:

NC Type Name	Description	Sample	Remediation Factor
Type1	Corrigible with an automated tool, no risk	Change in the indentation	0.01
Type2	Manual remediation, but no impact on compilation	Add some comments	0.1
Type3	Local impact, need only unit testing	Replace an instruction by another	1
Type4	Medium impact, need integration testing	Cut a big function in two	5
Type5	Large impact, need a complete validation	Change within the architecture	20

SQALE – Non-Remediation Function

- Non-remediation funtions represent the cost to keep a non-conformity so a negative impact from the business point of view

NC Type	Description	Sample	Non-Remediation Factor
Blocking	Will or may result in a bug	Division by zero	5 000
High	Wil have a high/direct impact on the maintainance cost	Copy and paste	250
Medium	Will have a medium/potential impact on the maintainance cost	Complex logic	50
Low	Wil have a low impact on the maintainance cost	Naming convention	15
Report	Very low impact, it is just a remediation cost report	Presentation issue	2

SQALE – Indices

- Sums of all the remediation costs associated to a particular hierarchy of characteristics constitute an index:
  - SQALE Testability Index: STI
  - SQALE Reliability Index: SRI
  - SQALE Changeability Index: SCI
  - SQALE Efficiency Index: SEI
  - SQALE Security Index: SSI
  - SQALE Maintainability Index: SMI
  - SQALE Portability Index: SPI
  - SQALE Reusability Index: SRuI
  - SQALE Quality Index: SQI (overall index)

* Note that there is a version of each index that represents density, normalized by some measure of size

SQALE – Rating

- Indexes can be used to build a rating value:

$$Rating = \frac{estimated\ remediation\ cost}{estimated\ development\ cost}$$

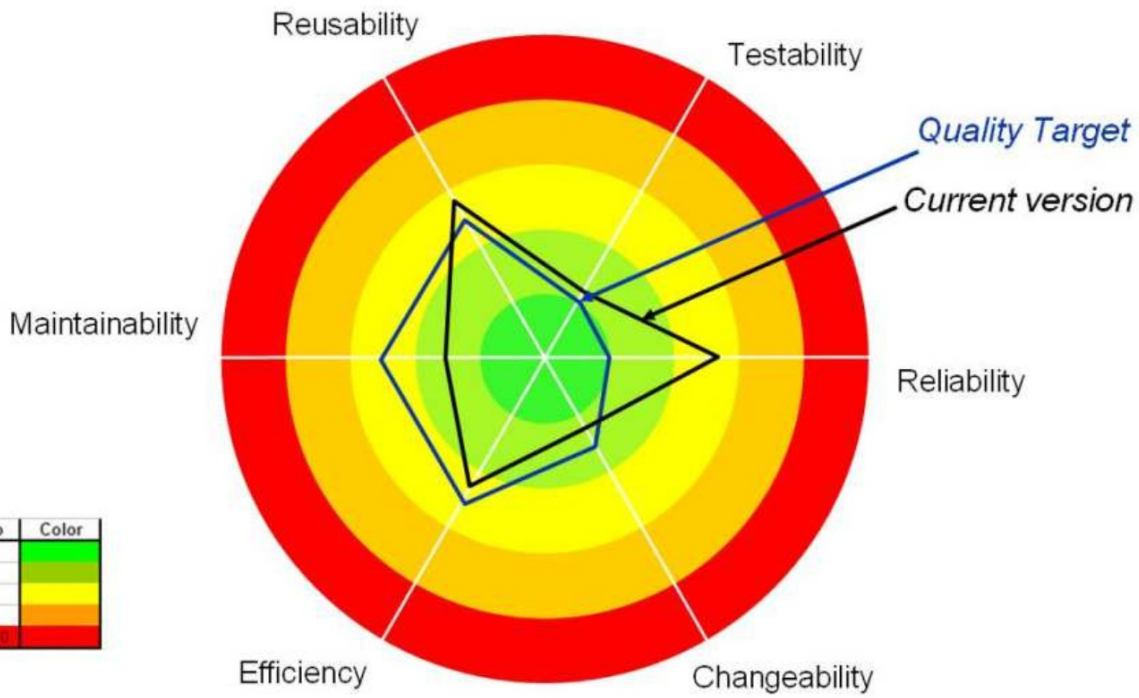
Rating	Up to	Color
A	1%	
B	2%	
C	4%	
D	8%	
E	∞	

Example, an artefact that has an estimated development cost of 300 hours and a STI of 8.30 hours, and using the reference table on the left

$$Rating = \frac{8.30\,h}{300\,h} = 2.7\,\% \text{->} C$$

SQALE – Rating

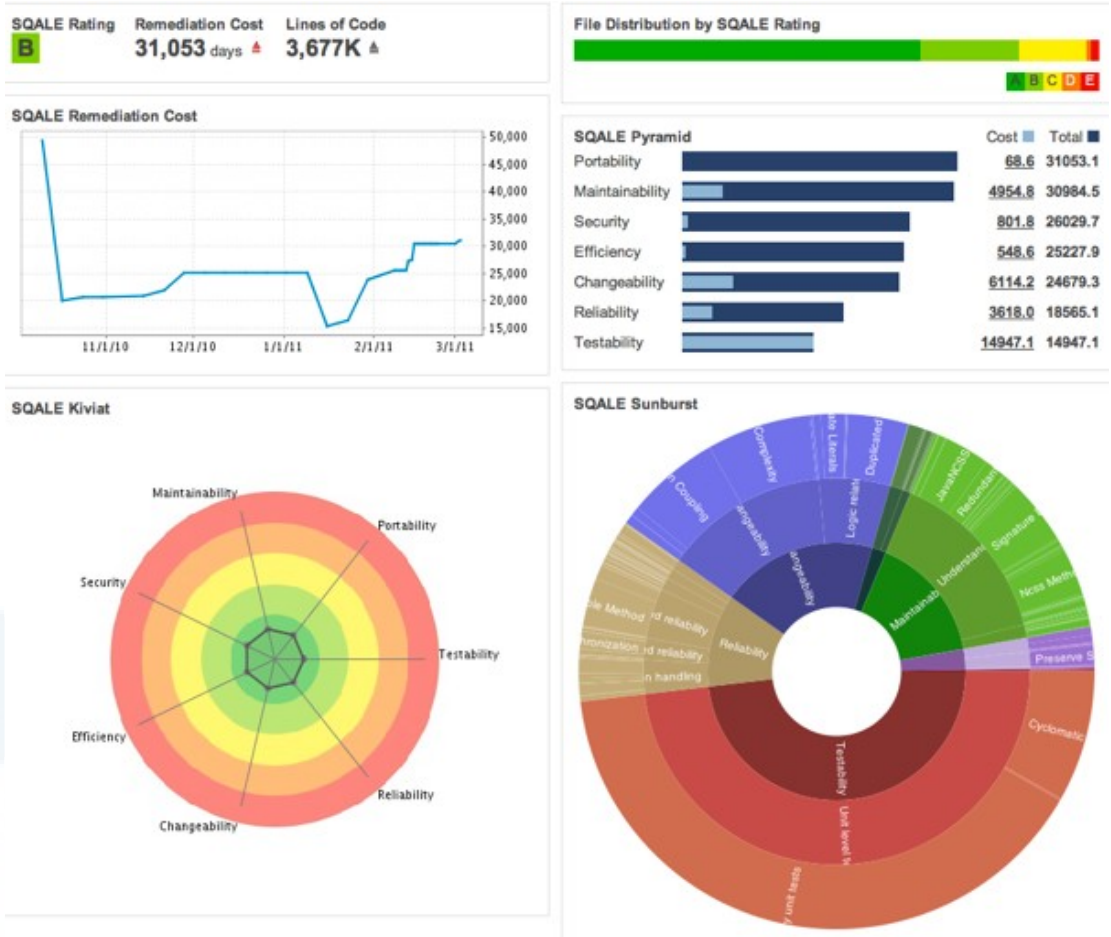
- The final representation can take the form of a Kiviat diagram in which the different density indexes are represented



Rating	UpTo	Color
A	0.3	
B	1	
C	3	
D	10	
E	over 10	

SQALE – Rating

- This is the view you find in SonarCube
  http://www.sonarqube.org/sonar-sqale-1-2-in-screenshot



SQALE

- Given our initial discussion of measurement pitfalls, scales and representation condition, the following sentence should be now clear:

  *"Because the non-remediation costs are **not** established on an **ordinal scale but on a ratio scale**, we have shown [..] that we can **aggregate the measures by addition and comply with the measurement theory and the representation clause**."*

  Letouzey, Jean-Louis, and Michel Ilkiewicz. "Managing technical debt with the SQALE method." IEEE software 6 (2012): 44-51.

Case Studies

Case Study

- Suppose that we have the some projects on which we computed the following set of metrics

	Project01	Project02	Project03	Project04	Project05	Project06
# LOCS	4920	5817	4013	4515	3263	5735
# packages	29	49	33	35	25	33
# classes	126	199	159	181	75	198
# methods	658	862	644	817	415	715
# attributes	153	196	227	285	78	177
# parameters	301	459	393	440	182	415
# local vars	493	533	325	397	339	416
# calls	2051	2830	1844	2297	917	2015
Proj_status	complete	complete	incomplete	complete	incomplete	complete

- What can you say about the projects?

Case Study

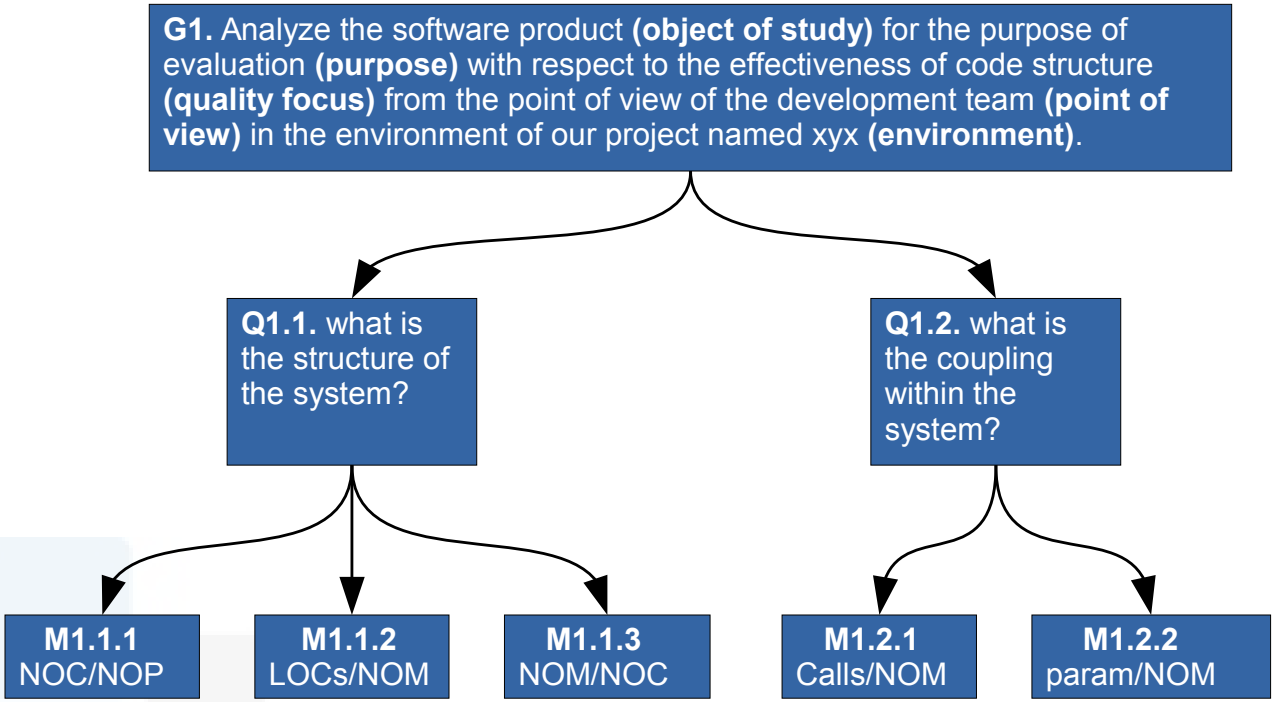
- What if we consider **relative** instead of **absolute values**?
- This would allow to compare the values across projects

	Project01	Project02	Project03	Project04	Project05	Project06
LOCs/NOM	7.48	6.75	6.23	**5.53**	7.86	**8.02**
NOC/NOP	4.34	4.06	4.82	5.17	**3.00**	**6.00**
NOM/NOC	5.22	4.33	4.05	4.51	**5.53**	**3.61**
att/NOC	1.21	0.98	1.43	**1.57**	1.04	**0.89**
param/NOM	0.46	0.53	**0.61**	0.54	**0.44**	0.58
locvars/NOM	0.75	0.62	0.50	**0.49**	**0.82**	0.58
Calls/NOM	3.12	**3.28**	2.86	2.81	**2.21**	2.82
Proj_status	**complete**	**complete**	**incomplete**	**complete**	**incomplete**	**complete**

highest value
lowest value

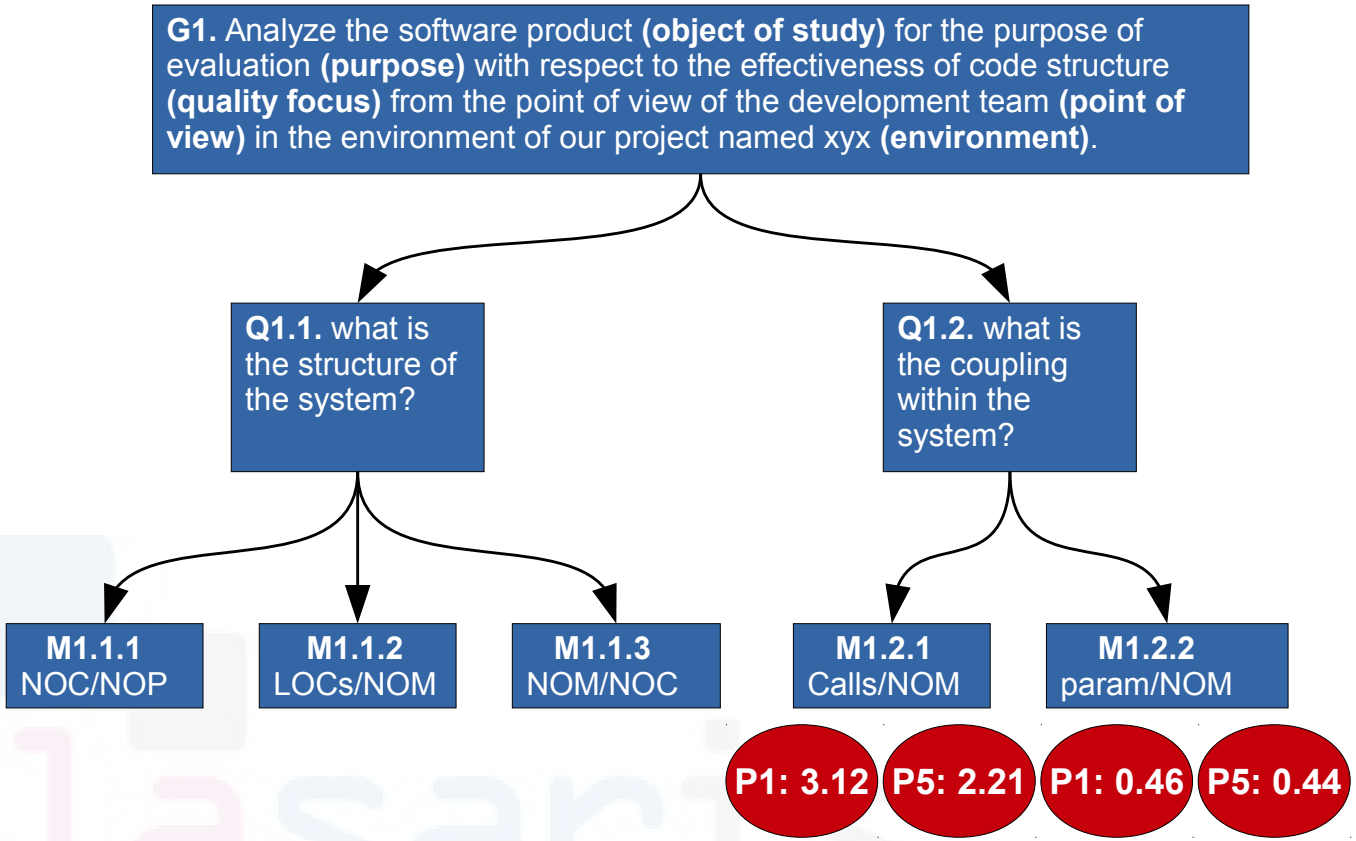
Case Study

- What if we make sense out of the metrics by using the GQM approach?



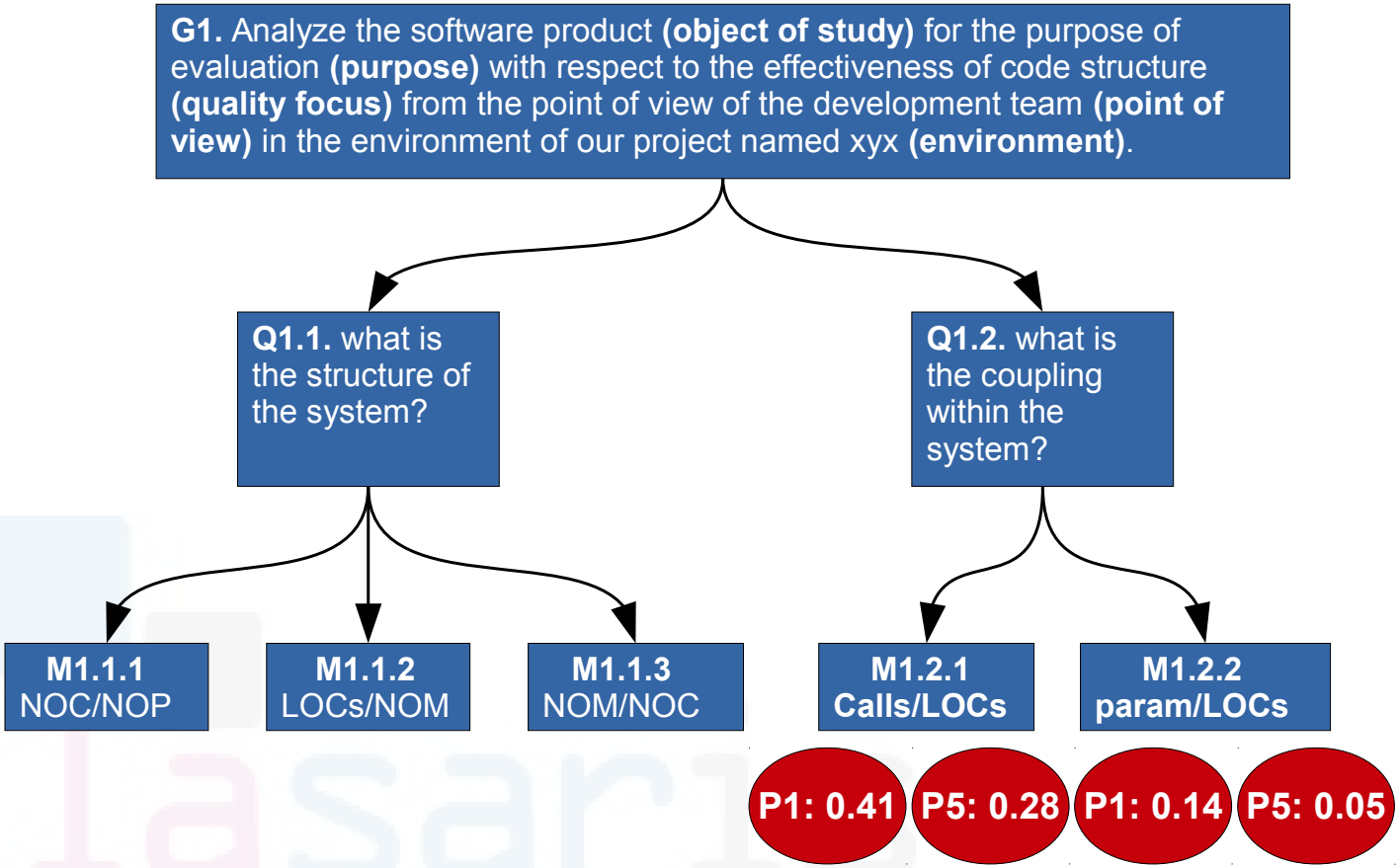
Case Study

- What if we make sense out of the metrics by using the GQM approach?



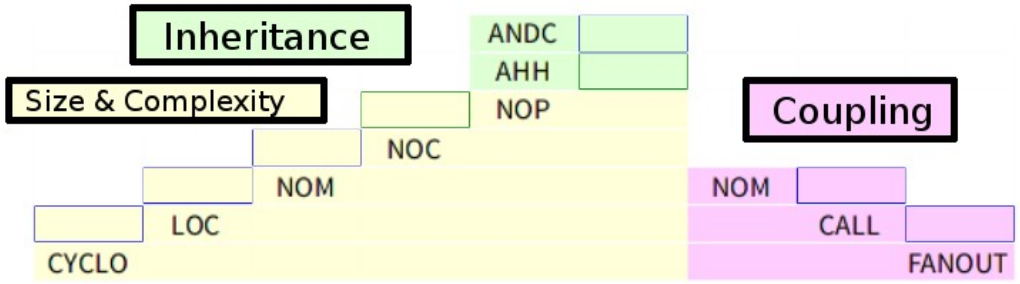
Case Study

- What happens if we consider LOCs instead of NOMs?



Case Study – The Overview Pyramid

- Another useful way to think in terms of relative values and thresholds is to use the Overview Pyramid
- The Overview pyramid allows to represent three different aspects of internal quality: inheritance, size & complexity and coupling



NOP: Number of Packages
NOC: Number of Classes
NOM: Number of Methods
LOC: Lines of Code
CYCLO: Cyclomatic Complexity

ANDC: Average Number of Derived Classes
AHH: Average Hierarchy Height
CALL: Number of Distinct Method Invocations
FANOUT: Number of Called Classes

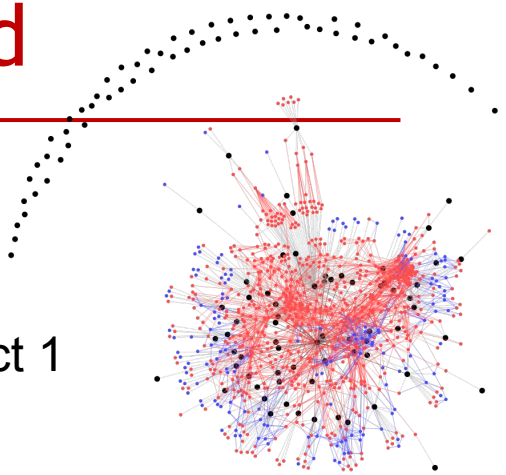
- It provides both absolute and relative values that are compared against typical thresholds

Case Study - The Overview Pyramid

Close to high
Close to average
Close to low

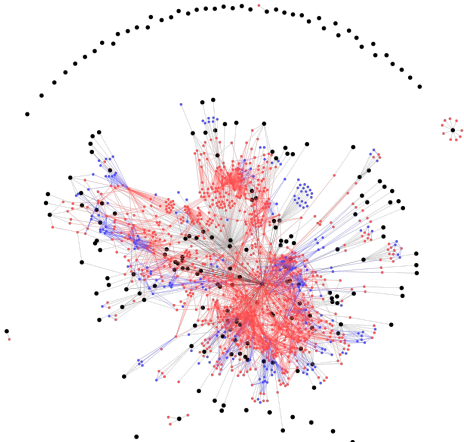
				ANDC	0.04			
				AHH	0.2			
			16.21	NOP	14			
		2.89		NOC	227			
	7.47		NOM		658	NOM	2.13	
0.1		LOC			4920.0	1407	CALL	0.13
	CYCLO				498.0	188		FANOUT

Project 1



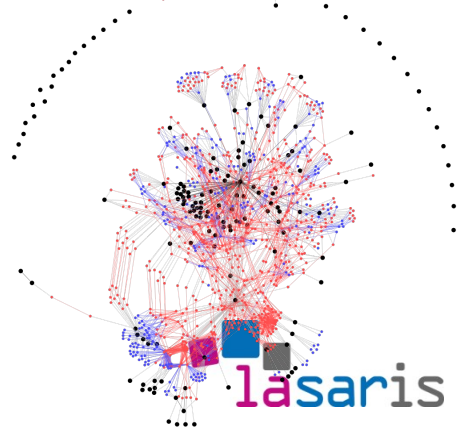
				ANDC	0.07			
				AHH	0.28			
			10.04	NOP	23			
		3.73		NOC	231			
	6.74		NOM		862	NOM	2.43	
0.11		LOC			5817.0	2098	CALL	0.16
	CYCLO				674.0	336		FANOUT

Project 2



				ANDC	0.05			
				AHH	0.28			
			11.25	NOP	16			
		3.57		NOC	180			
	6.23		NOM		644	NOM	1.86	
0.11		LOC			4013.0	1200	CALL	0.24
	CYCLO				456.0	299		FANOUT

Project 3

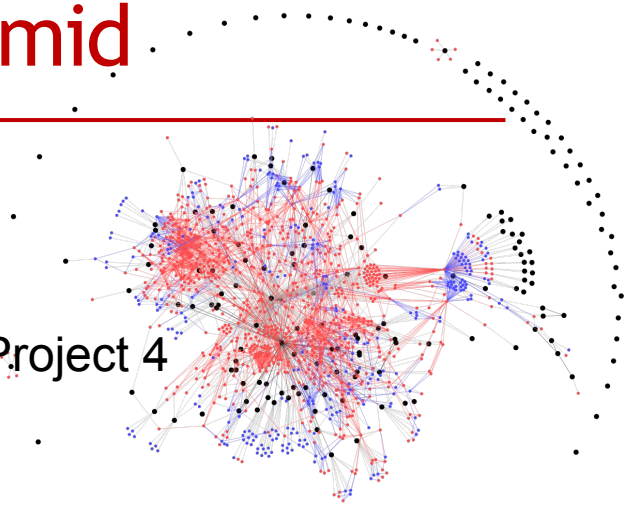


Case Study - The Overview Pyramid

Close to high
Close to average
Close to low

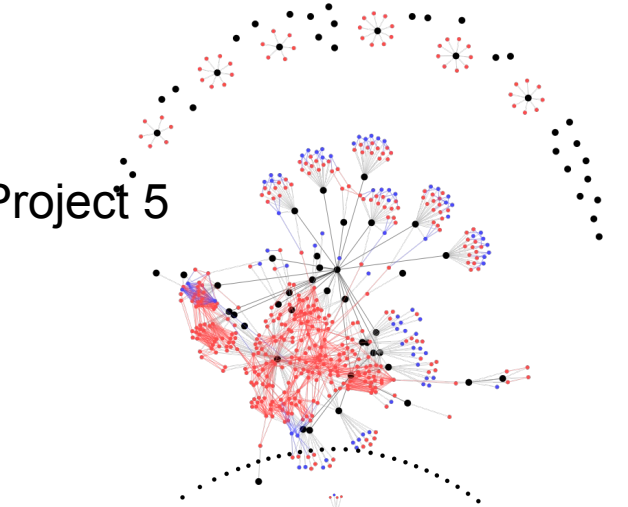
			ANDC	0.07		
			AHH	0.28		
		14.0	NOP	17		
		3.43	NOC	238		
	5.52	NOM		817	NOM	1.92
0.11	LOC			4515.0	1574	CALL	0.21
CYCLO				535.0	331		FANOUT

Project 4



			ANDC	0.11		
			AHH	0.59		
		5.63	NOP	11		
		6.69	NOC	62		
	7.86	NOM		415	NOM	1.67	
0.1	LOC			3263.0	697	CALL	0.11
CYCLO				351.0	80		FANOUT

Project 5



			ANDC	0.06		
			AHH	0.22		
		14.25	NOP	16		
		3.13	NOC	228		
	8.02	NOM		715	NOM	2.0	
0.13	LOC			5735.0	1431	CALL	0.17
CYCLO				757.0	257		FANOUT

Project 6

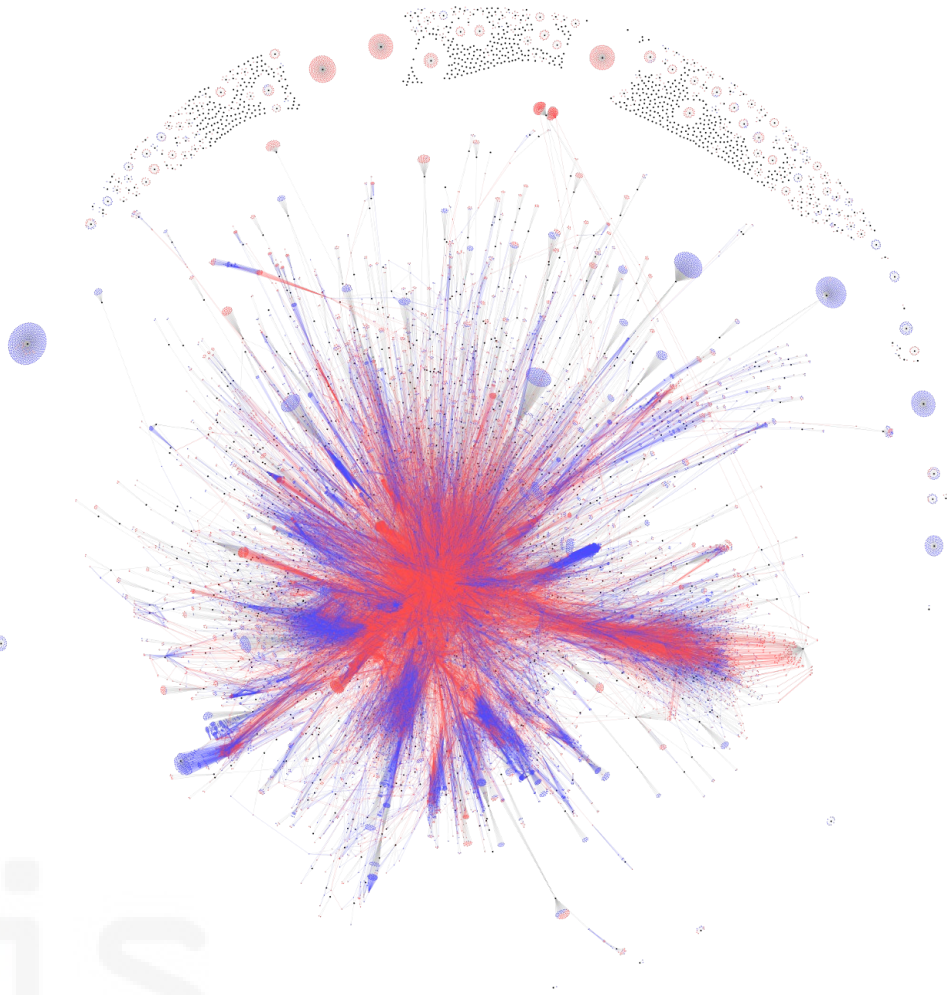
Case Study – The Overview Pyramid

Back to our initial project
Eclipse JDT 3.5.0

The overview pyramid

				ANDC	0.0			
				AHH	0.0			
			29.42	NOP	45			
		17.82	NOC		1324			
	15.02	NOM			23605	NOM	2.19	
0.2	LOC				354780.0	51765	CALL	0.07
CYCLO					72883.0	3804		FANOUT

Close to high
Close to average
Close to low



Conclusions

- Measurement is important to track progress of software projects and to focus on relevant parts that need attention
- As such, we always need to take measurement into account with some *"grain of salt"*
- Still, collecting non-relevant or non-valid metrics might be even worse than not collecting any valid measure at all

Extra Slides

List of some Acronyms

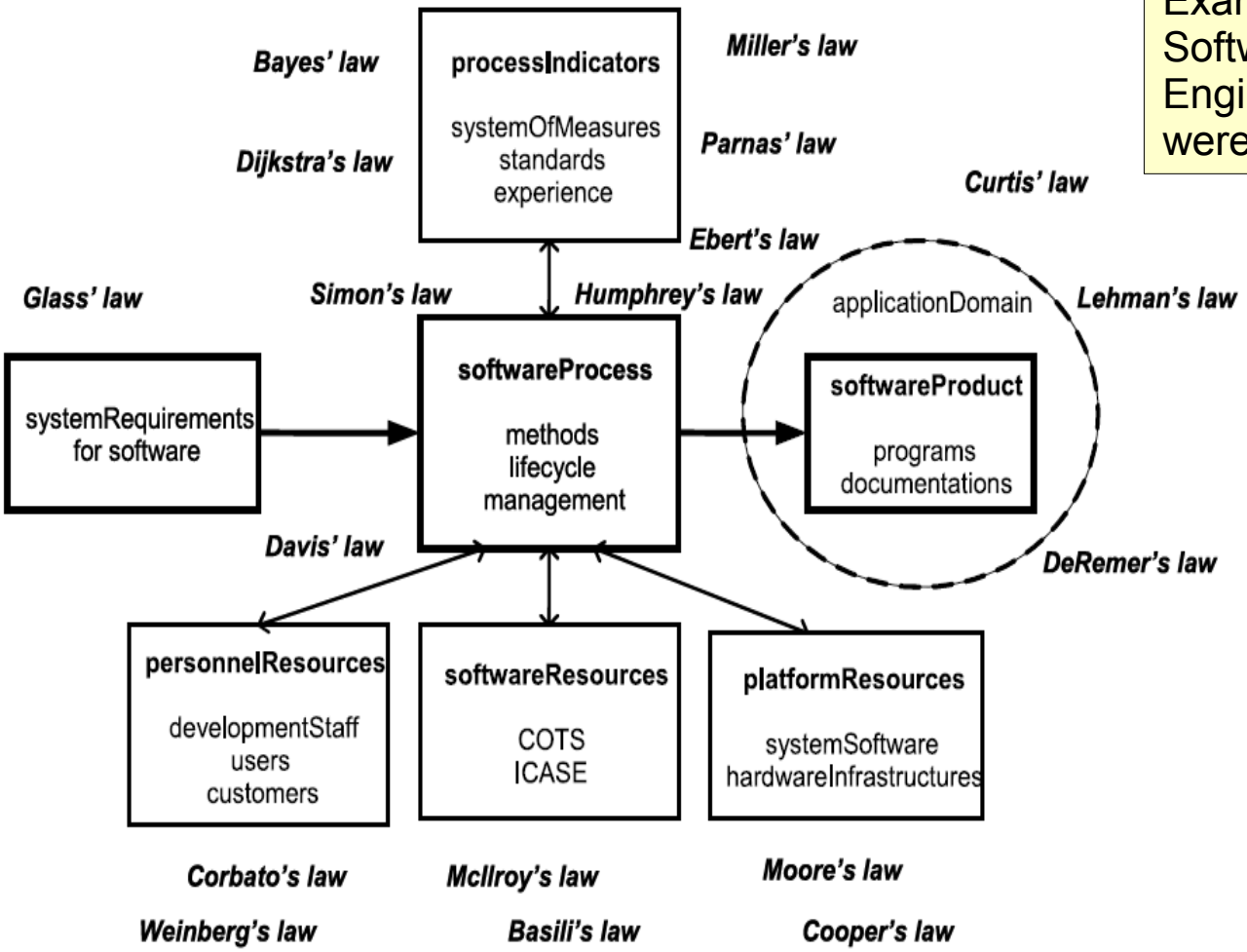
- LOCs: Lines of Code
- CC: McCabe Cyclomatic complexity
- Fan in: number of local flows that terminates in a module
- Fan out: number of local flows emanate from a module
- Information flow complexity of a a module: length of the module times the squared difference of fan in and fan out
- NOM: Number of Methods per class
- WMC: Weighted Methods per Class
- DIT: Depth of Inheritance Tree
- NOC: Number of Children
- CBO: Coupling Between Objects
- RFC: Response For a Class
- LCOM: Lack of Cohesion of Methods
- ANDC: Average Number of Derived Classes
- AHH: Average Hierarchy Height

Measurement Experience

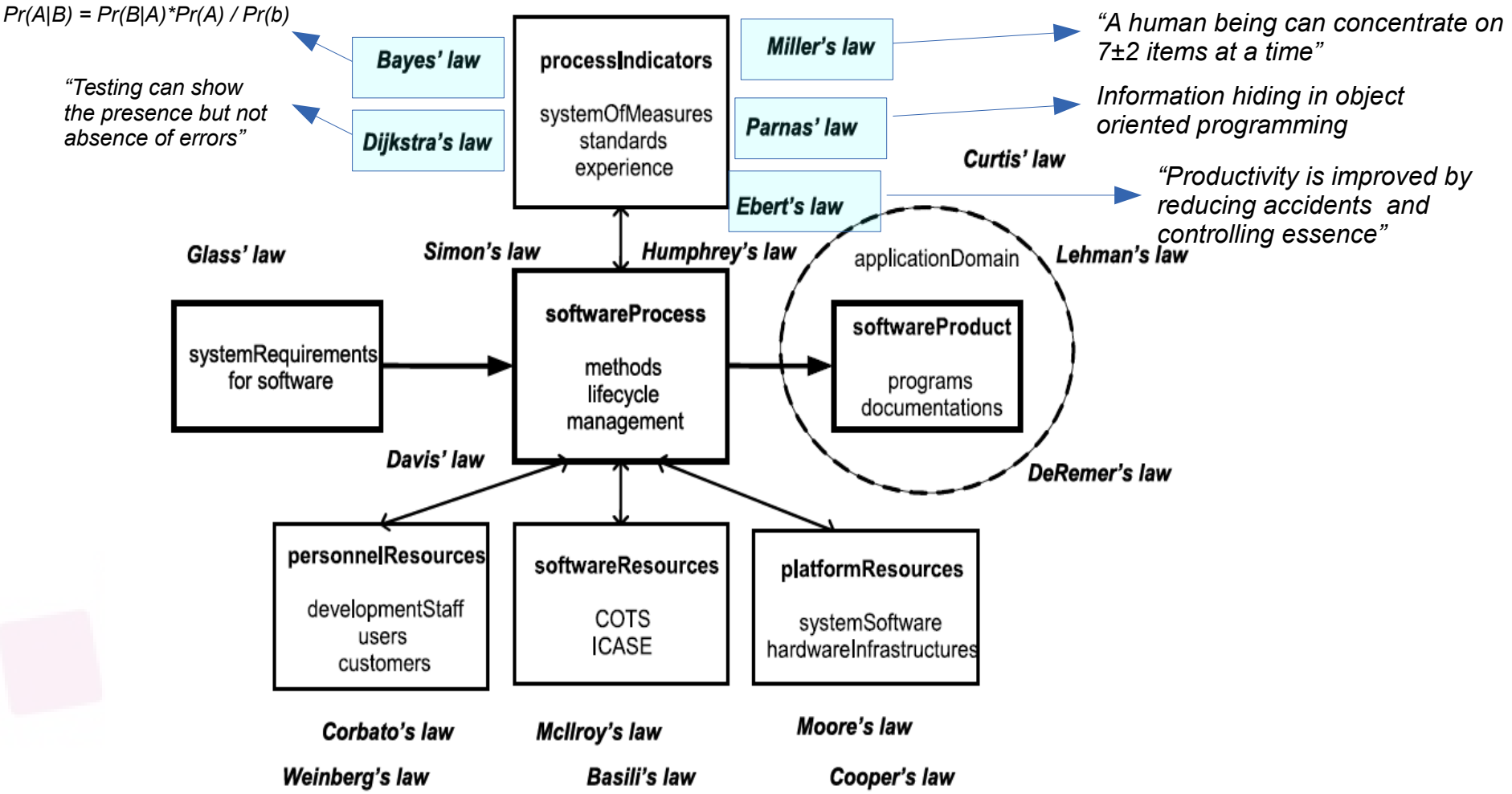
- **Measurement Experience** can have the form of:
  - Analogies
  - Axioms
  - Correlations
  - Criterions
  - Intuitions
  - Laws
  - Lemmas
  - Formulas,
  - Methodologies
  - Principles
  - Relations
  - Rule Of Thumbs
  - Theories

Software Engineering Laws (1/4)

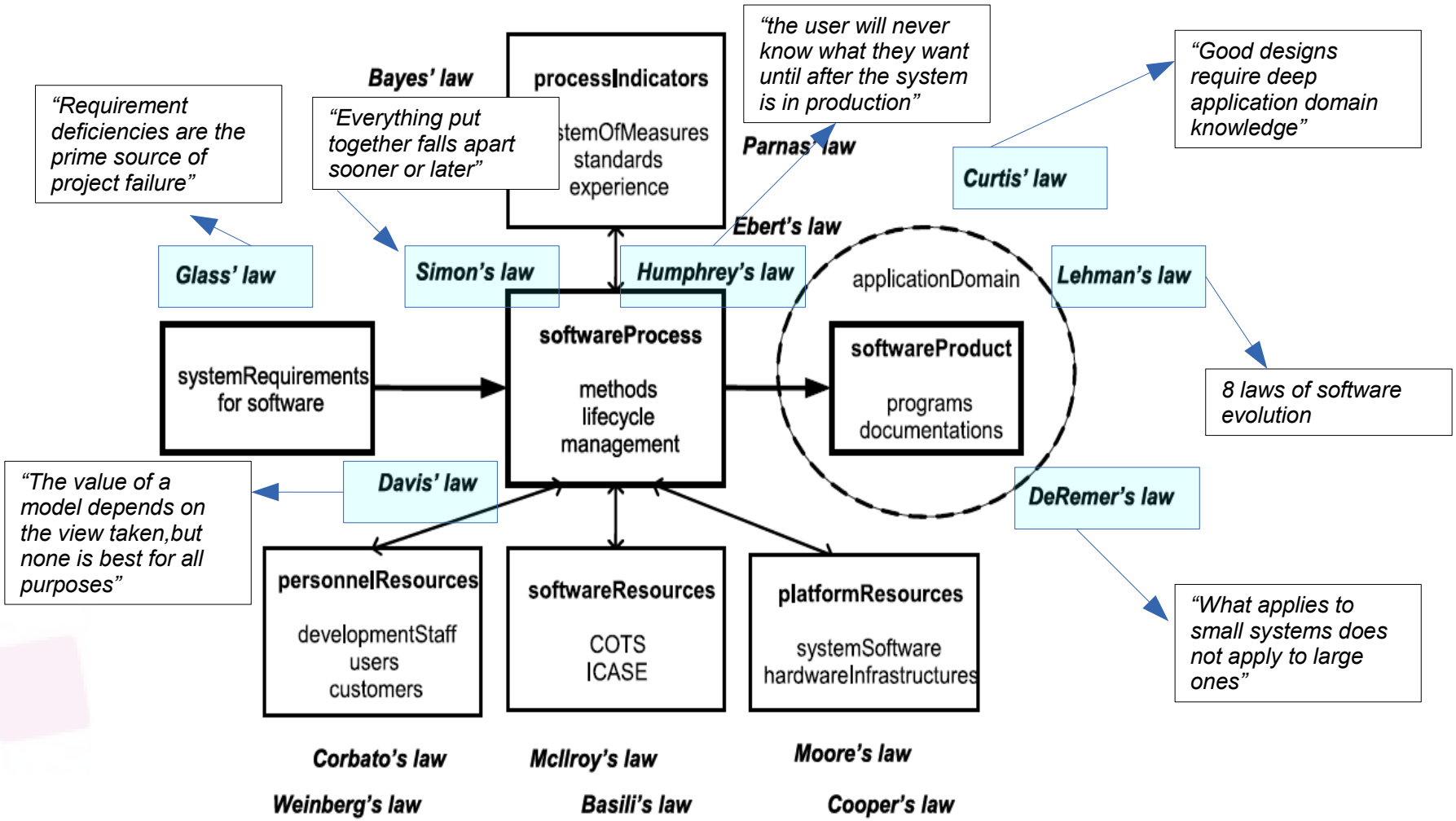
Example: Laws in Software Engineering: how were these derived?



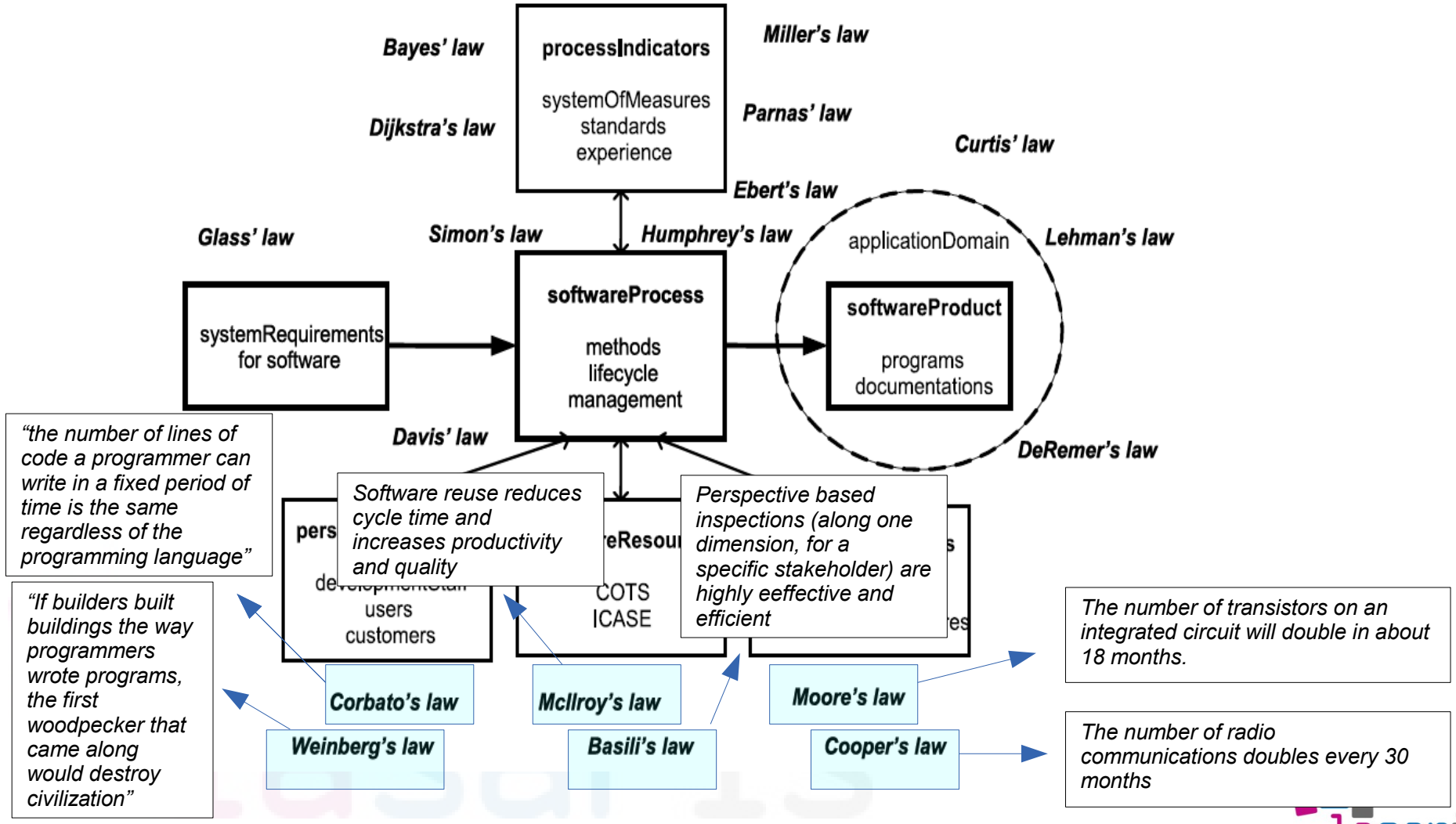
Software Engineering Laws (2/4)



Software Engineering Laws (3/4)

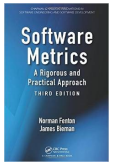


Software Engineering Laws (4/4)



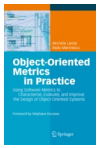
References

- N. Fenton and J. Bieman, Software Metrics: A Rigorous and Practical Approach, Third Edition, 3 edition. Boca Raton: CRC Press, 2014.



- C. Ebert and R. Dumke, Software Measurement: Establish - Extract - Evaluate - Execute, Softcover reprint of hardcover 1st ed. 2007 edition. Springer, 2010.

- Lanza, Michele, and Radu Marinescu. Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media, 2007.



- Some code samples from Martin, Robert C. Clean code: a handbook of agile software craftsmanship. Pearson Education, 2008.

- Moose platform for software data analysis http://moosetechnology.org

- The SQALE Method http://www.sqale.org/wp-content/uploads/2010/08/SQALE-Method-EN-V1-0.pdf