PV260 - SOFTWARE QUALITY

[Spring 2024]

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

Introduction

- The following defect (*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

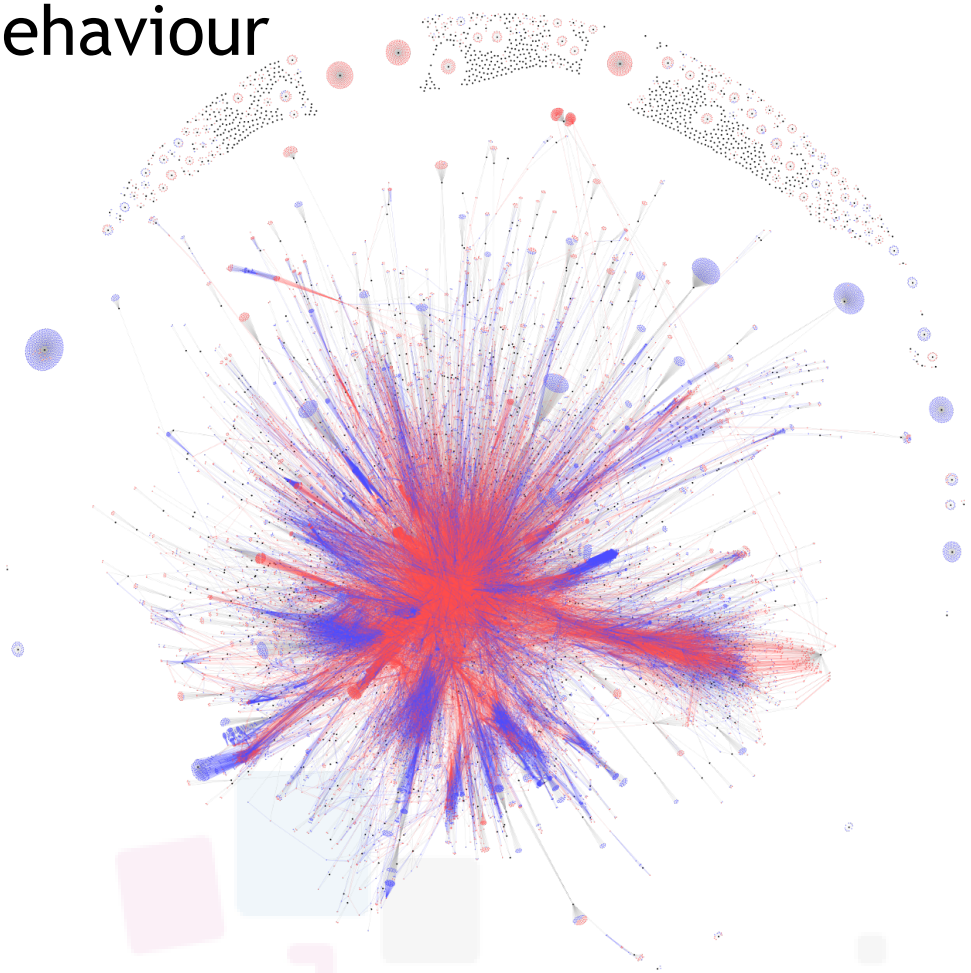
The duplicate handshake algorithm appearing immediately before the buggy block

```
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

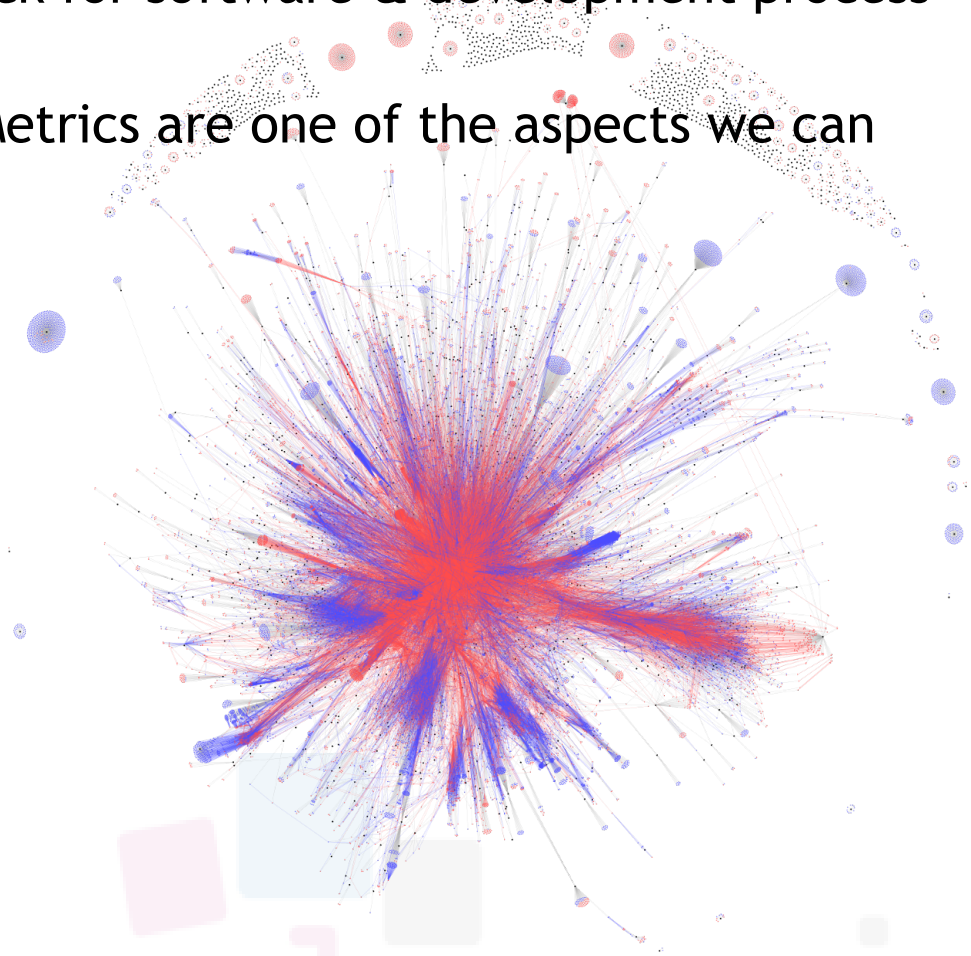
- Modern systems are very large & complex in terms of structure & runtime behaviour
- The figure on the right represents Eclipse JDT 3.5.0 (350K LOCs, 1.324 classes, 23.605 methods)



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?
What are these metrics “good” for?

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

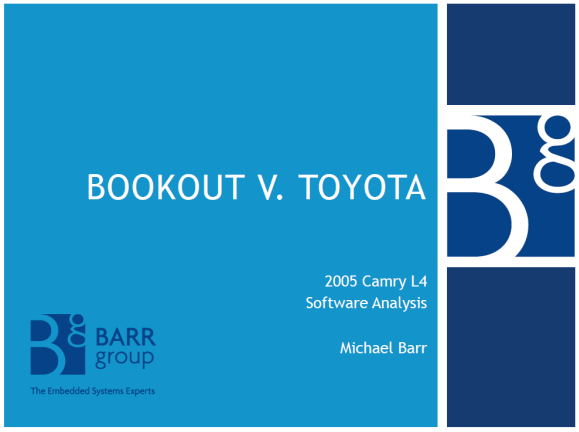
Introduction

- Typical problems related to software measurement:
  - → **How can I measure the maintainability of my software?**
  - → **Can I estimate the number of defects of my software?**
  - → **What is the productivity of my development team?**
  - → **Can I measure the quality of my testing process?**

Motivational Example

Review of defective Toyota Camry's System (1/3)

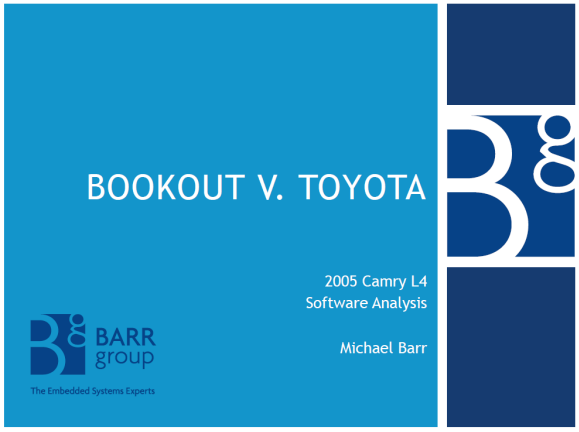
- Expert source code and system review after reported cases of accidents due to cars accelerating without users' inputs *
- 18 months review + previous NASA experts code review
- Investigation on unintended accelerations



* http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf

Review of defective Toyota Camry's System (2/3)

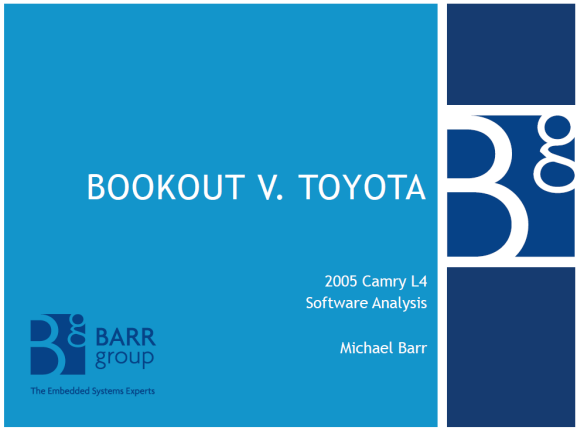
- Usage of software metrics (p.24):
- "Data-flow spaghetti
  - Complex coupling between software modules and between tasks
  - Count of global variables is a software metric for "tangledness"
    → 2005 Camry L4 has >**11,000** global variables (NASA)"



* http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf

Review of defective Toyota Camry's System (3/3)

- Usage of software metrics (p.24):
- "Control-flow spaghetti
  - Many long, overly-complex function bodies
  - Cyclomatic Complexity is a software metric for "testability"
    → 2005 Camry L4 has **67 functions scoring >50** ("untestable")
    → The throttle angle function **scored over 100** (unmaintainable)"
- See also **p.30-31** for coding rules violations and expected number of bugs



* http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf

Background on Software Measurement

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** (Fenton & Pfleeger, 1997)

Why Software Measurement

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

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

However…

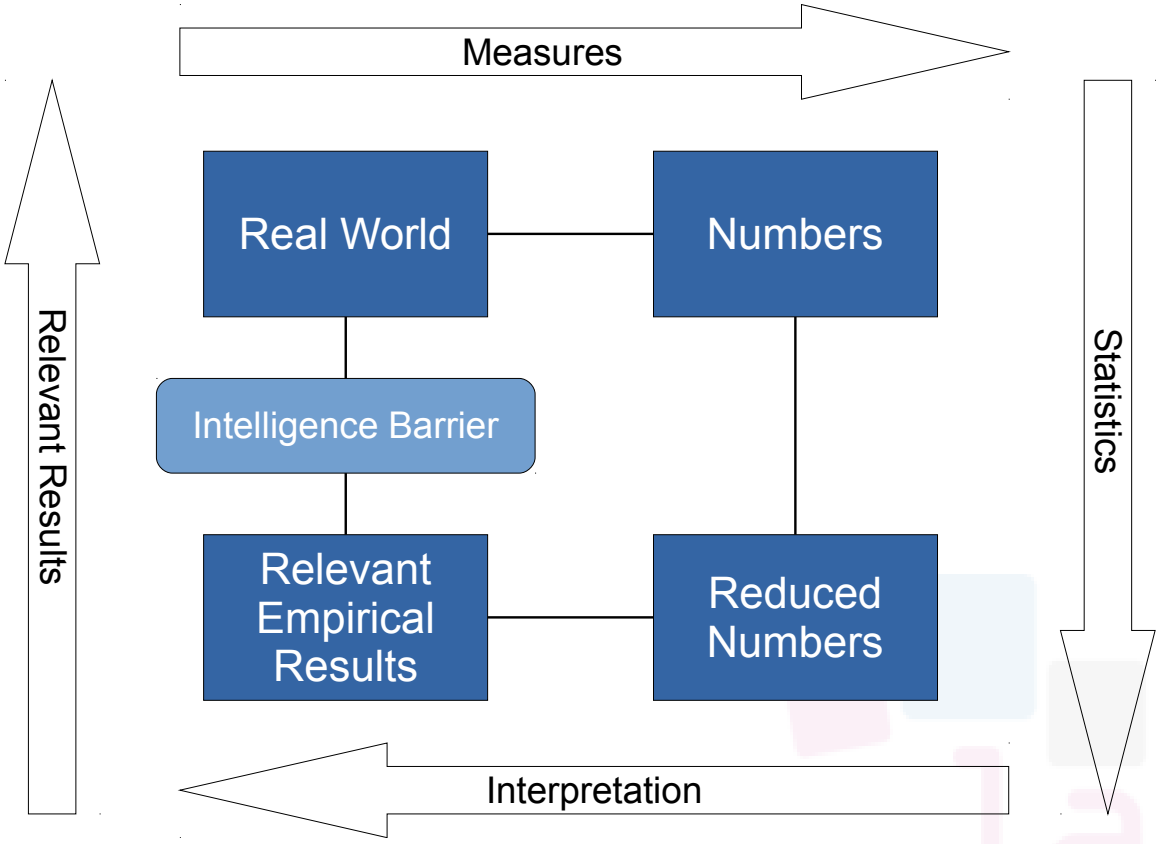
- Although measurement may be integrated in development, very often **the objectives** of measurements are not clear

  "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

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**



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**

Real World \ Measurement	Low	High
Low	TRUE NEGATIVE	FALSE POSITIVE
High	FALSE NEGATIVE	TRUE POSITIVE

- **Example: Is LOC a valid measure of productivity?**

  → Think by paradox: 100K *system.out* statements vs 100K of complex loops and statements

Real World \ Measurement	Low	High
Low	TRUE NEGATIVE	FALSE POSITIVE
High	FALSE NEGATIVE	TRUE POSITIVE

ADDITIONAL PROBLEM: You might have two different projects with two different definitions of LOCs (e.g., considering blanks+comments vs only ";") so that the following can be true at the same time P1>P2 and P1<P2

Valid Measures - Example (1/5)

- **Code coverage** is a measure giving an indication of how much of the source code has been run ("covered") by running the tests
- Different criteria:
  - **Statement coverage** (the one assumed by standard "code coverage): the % of statements of the program covered by the tests
  - **Function coverage**: the % of functions/methods covered by the tests
  - **Branch coverage:** the % of branches of the control structures (e.g., if-→then-→else) covered by the tests
  - **Condition coverage:** % of each Boolean condition evaluated both as True/False

```
[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] }
```

Valid Measures – Example (2/5)

- From Wikipedia some years ago: *"...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..."* - *as of 2022 this sentence was removed from Wikipedia, but it is still in some other webpages (probably copy & paste)*

  Q.: 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 (3/5)

A. Assumption: considering 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);
```

→ Same coverage, but the one on the right is a better test

Note(!): Software follows usually a **Pareto principle:**

→ ~80% of the defects are in the ~20% of the code

→ the ~20% of code with more defect-density can be more difficult to cover with tests

Valid Measures – Example (4/5)

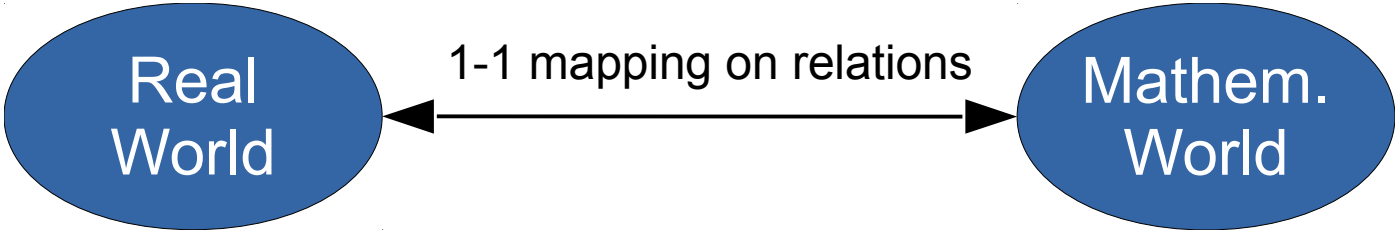
- According to **Martin Fowler**:

  → *"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 (5/5)

- In this case, **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



- You can see this also from the Information Theoretical point of view

Real World \ Measurement	Low	High
Low	TRUE NEGATIVE	FALSE POSITIVE
High	FALSE NEGATIVE	TRUE POSITIVE

Measurement Scales (1/4)

- Every measurement is mapped to a so-called scale (**nominal**, **ordinal**, **interval**, **rational**)
- Considering the scale is quite important for the admissible operations

	≠,=	<,>	min,max	median	avg	prop
Nominal →						
Ordinal →						
Interval →						
Rational →						

Measurement Scales (2/4)

- Some examples of measures and related scales

Scale Type	Examples in Software Eng.	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 (3/4) - example

- 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 (4/4) - example

- 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

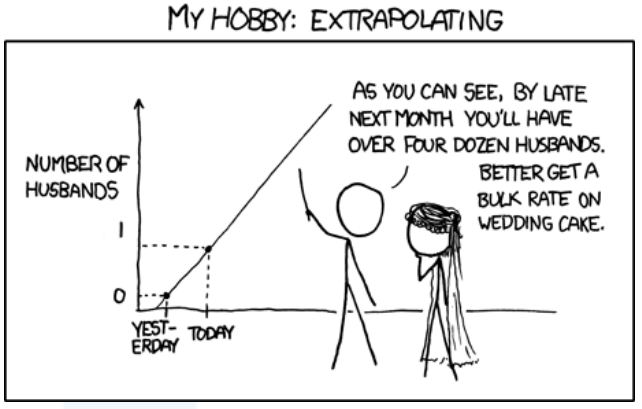
Let's define the following metric:

$$Sev(P_n) = avg(\sum issues_i * weight_i)$$

$$Sev(P_1) = avg(2*6+36*5+25*4+15*3+2*2+121*1) = 77$$
$$Sev(P_2) = avg(10*6+19*5+22*4+32*3+5*2+113*1) = 77$$

Are the projects the same according to our metric? Is there the "same distance" from a critical ticket to a blocker that there is between minor and trivial?

Pitfalls in linking the real world phenomenon to numbering systems



https://xkcd.com/605/

Pitfall Example (1/3) ⚠

- A/B Testing is a kind of **randomized experiment** in which you can propose **two variants** of the same application to the users
- We set-up an experiment with two browsers and two variations of the same webpage
- **Conversion Rate:** *% of users completing an action*

	Conv Rate A	Conv Rate B
Firefox	87.50%	100.00%
Chrome	50.00%	62.50%

What can you conclude? Which alternative is better?

https://medium.com/homeaway-tech-blog/simpsons-paradox-in-a-b-testing-93af7a2f3307

Pitfall Example (2/3) ⚠

- Let's look at the same table but with additional information about the way the tests were split

	Conv Rate A	Conv Rate B
Firefox	70/80 = 87.5%	20/20 = 100%
Chrome	10/20 = 50%	50/80 = 62.5%
Both	80/100 = 80%	70/100 = 70%

https://medium.com/homeaway-tech-blog/simpsons-paradox-in-a-b-testing-93af7a2f3307

Pitfall Example (3/3) ⚠

Simpsons' paradox

- It can happen that:

  `a/b < A/B`

  `c/d < C/D`

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

- example

  `1/5 (20%) < 2/8 (25%)`

  `6/8 (75%) < 4/5 (80%)`

  `7/13 (53%) > 6/13 (46%)`

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

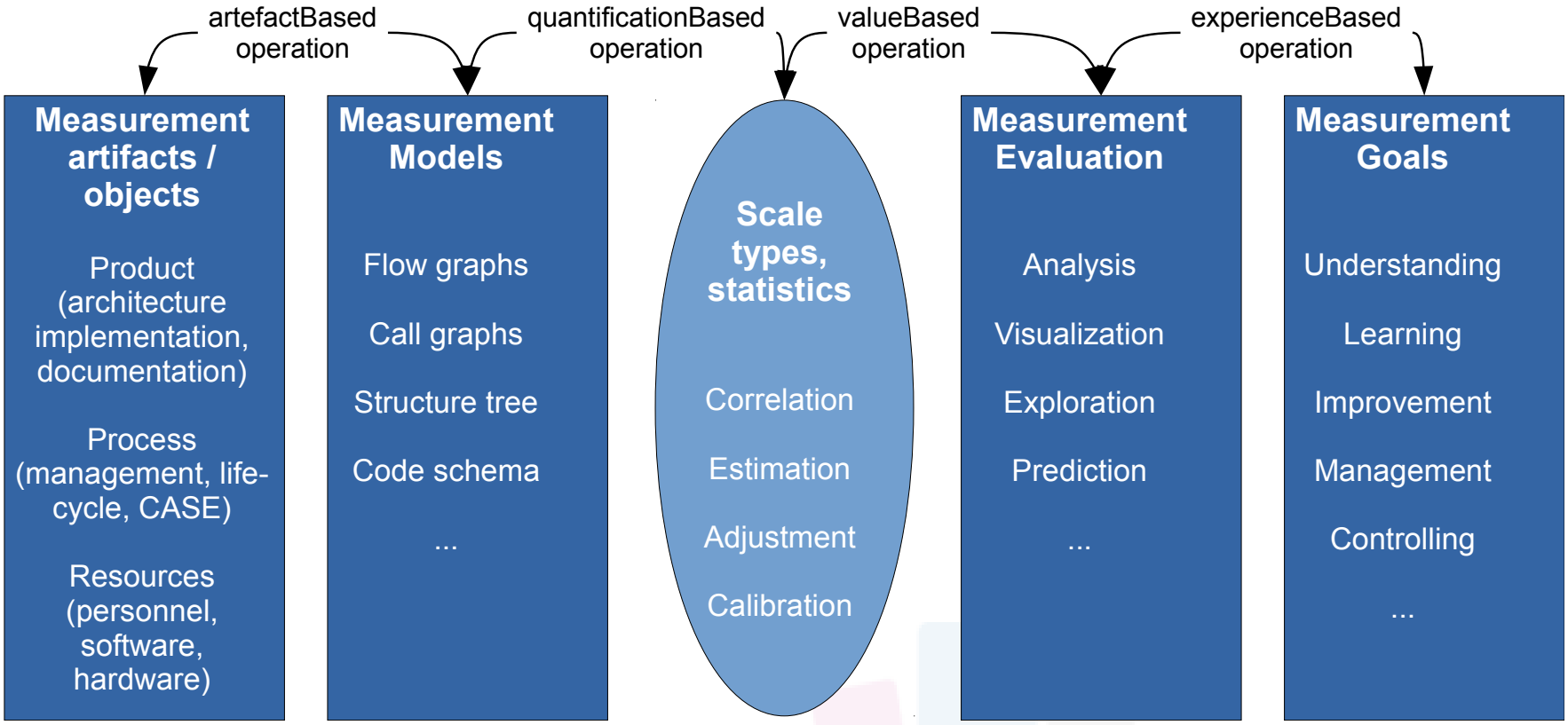
See: https://plato.stanford.edu/entries/paradox-simpson/ – considering the following papers:

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

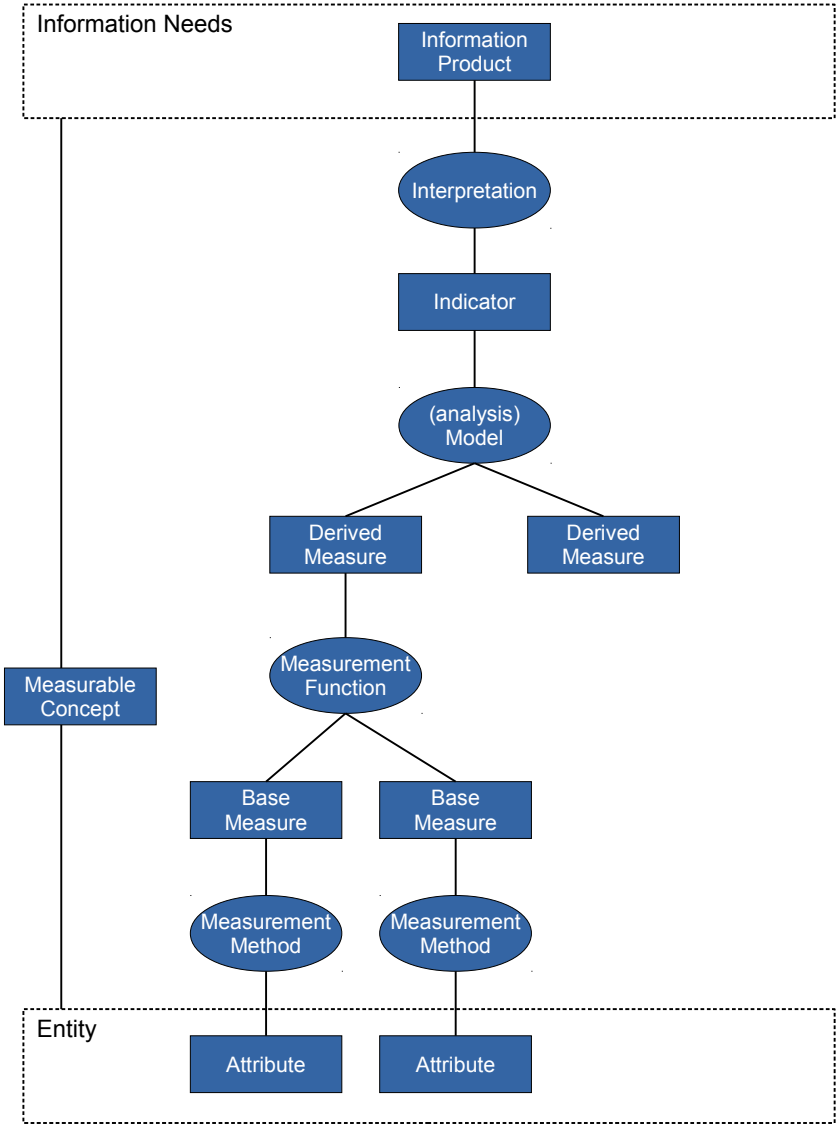
P.J. Bickel, E.A. Hammel and J.W. O'Connell (1975). "Sex Bias in Graduate Admissions: Data From Berkeley. Science 187 (4175): 398–40

Software Measurement Models & Methods

Software Measurement Methods



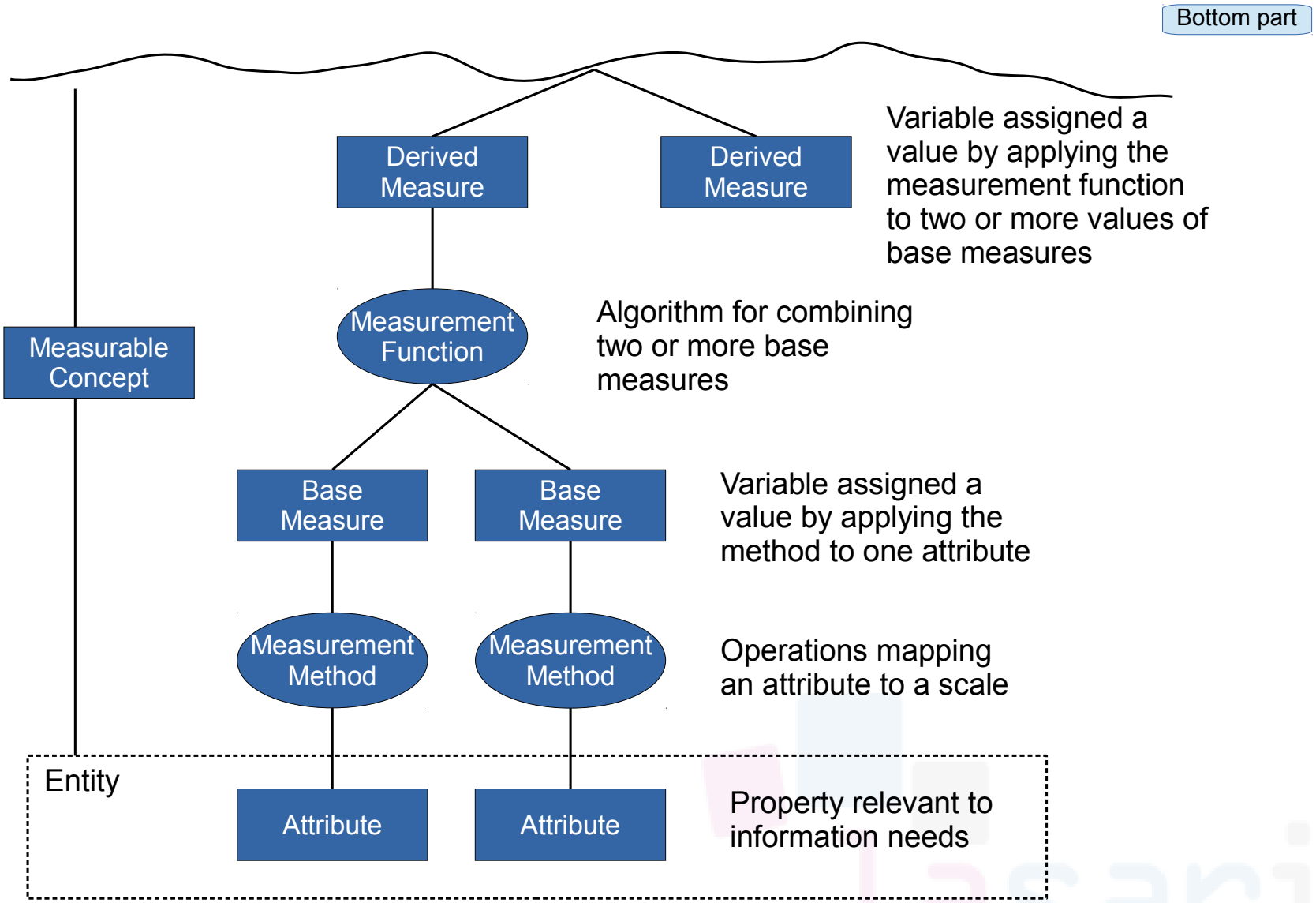
Measurement Information Model (ISO/IEC 15939)



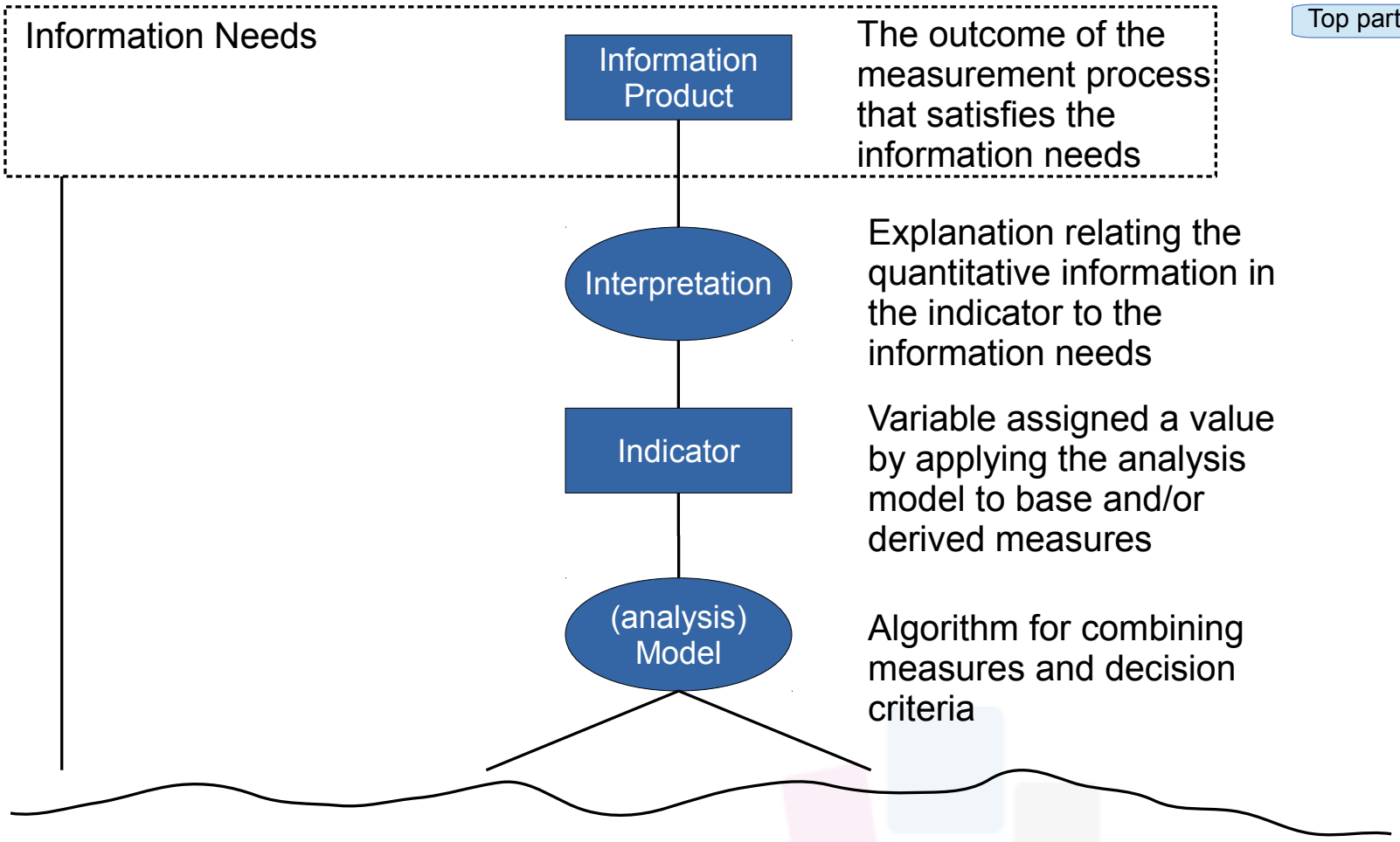
Measurable Concept:

abstract relationship between attributes of entities and information needs

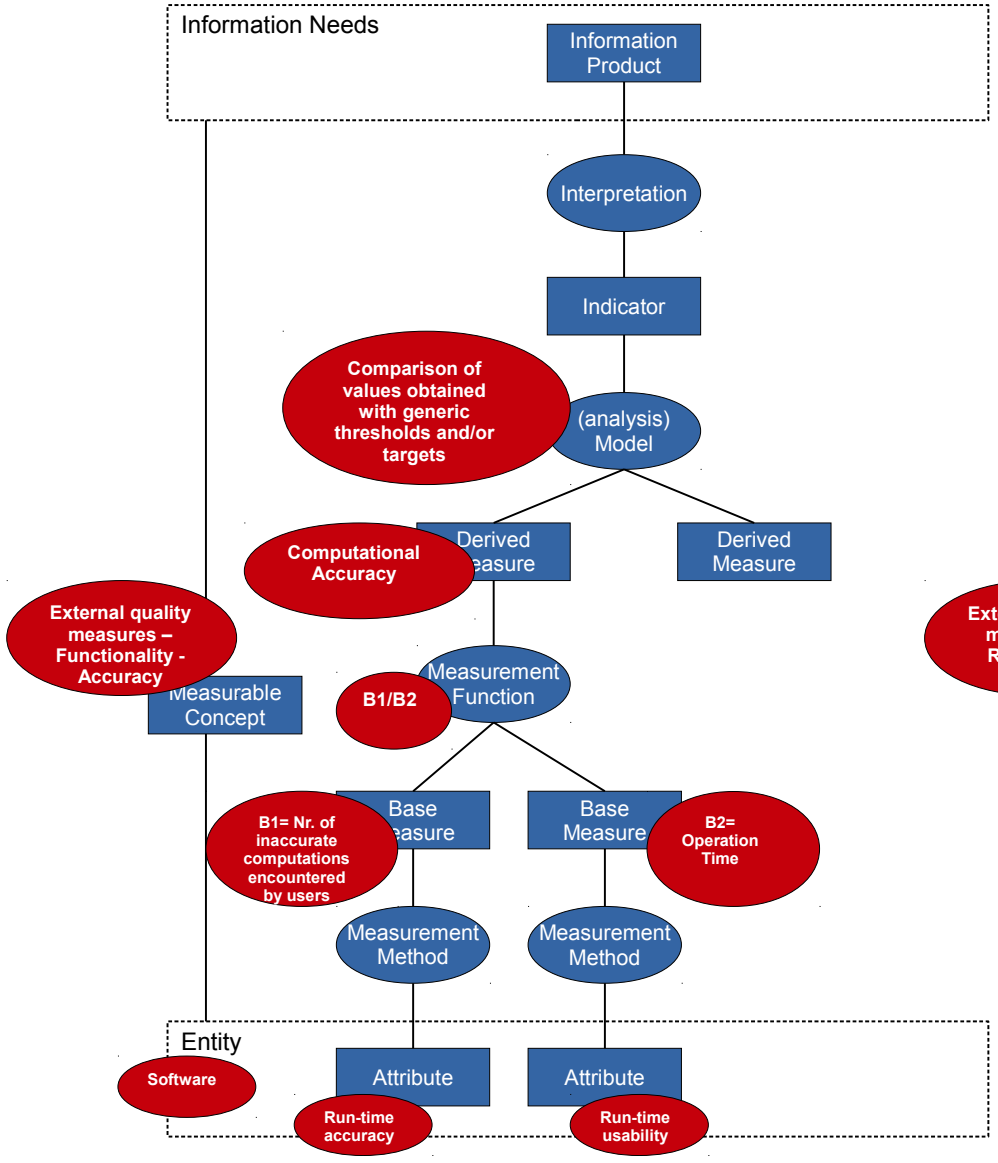
Measurement Information Model (ISO/IEC 15939)

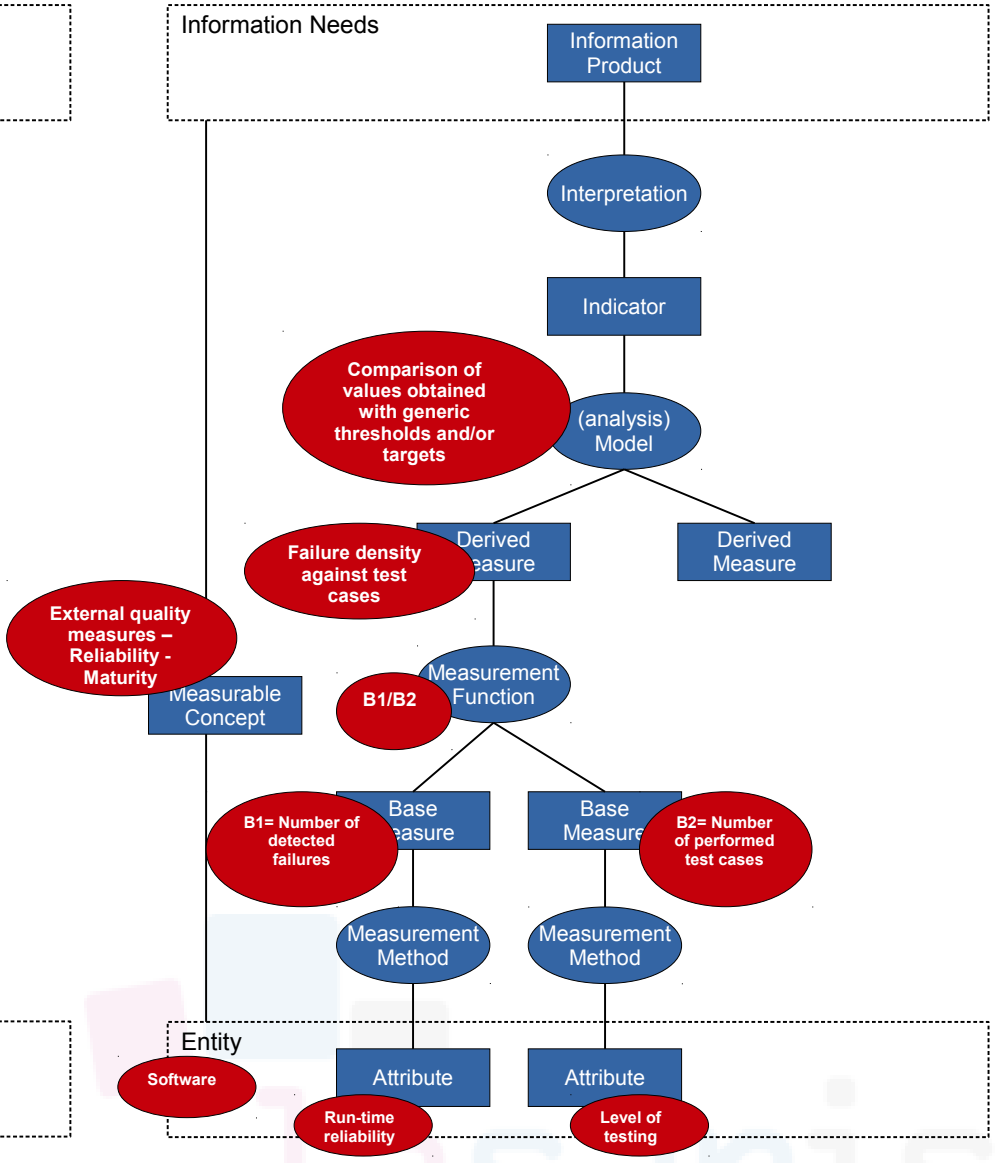


Measurement Information Model (ISO/IEC 15939)



ISO/IEC 15939 Examples

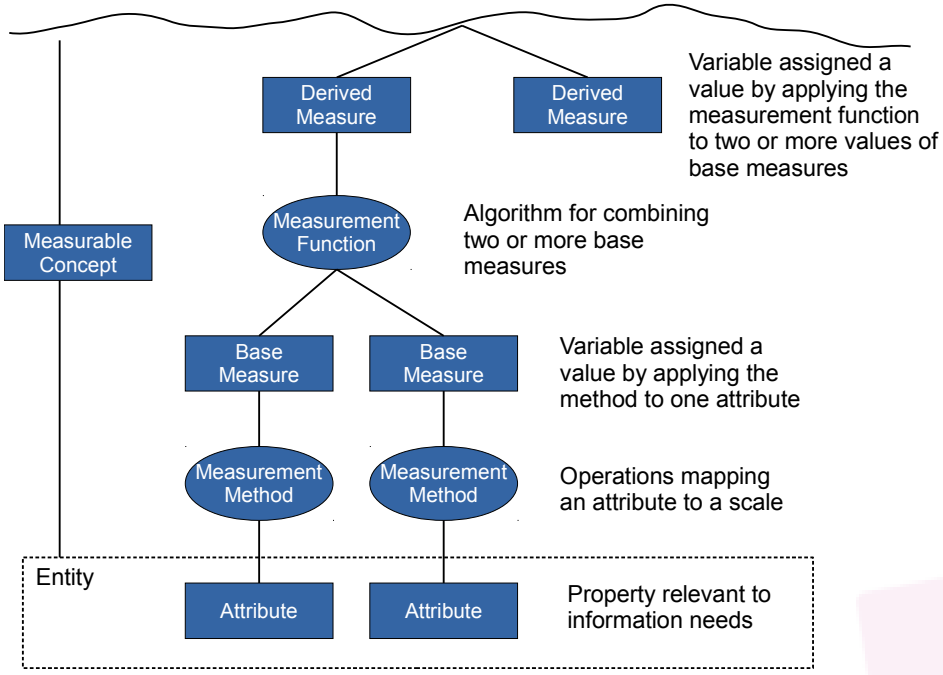




Inspired by Abran, Alain, et al. "An information model for software quality measurement with ISO standards." Proceedings of the International Conference on Software Development (SWDC-REK), Reykjavik, Iceland. 2005.

Direct vs Indirect Measures (1/2)

- 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



ISO/IEC 15939 refers to them as **base** measure and **derived** measure

Direct vs Indirect Measures (2/2)

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

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

  - COCOMO, measure of effort

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

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

$$a = 2.94$$

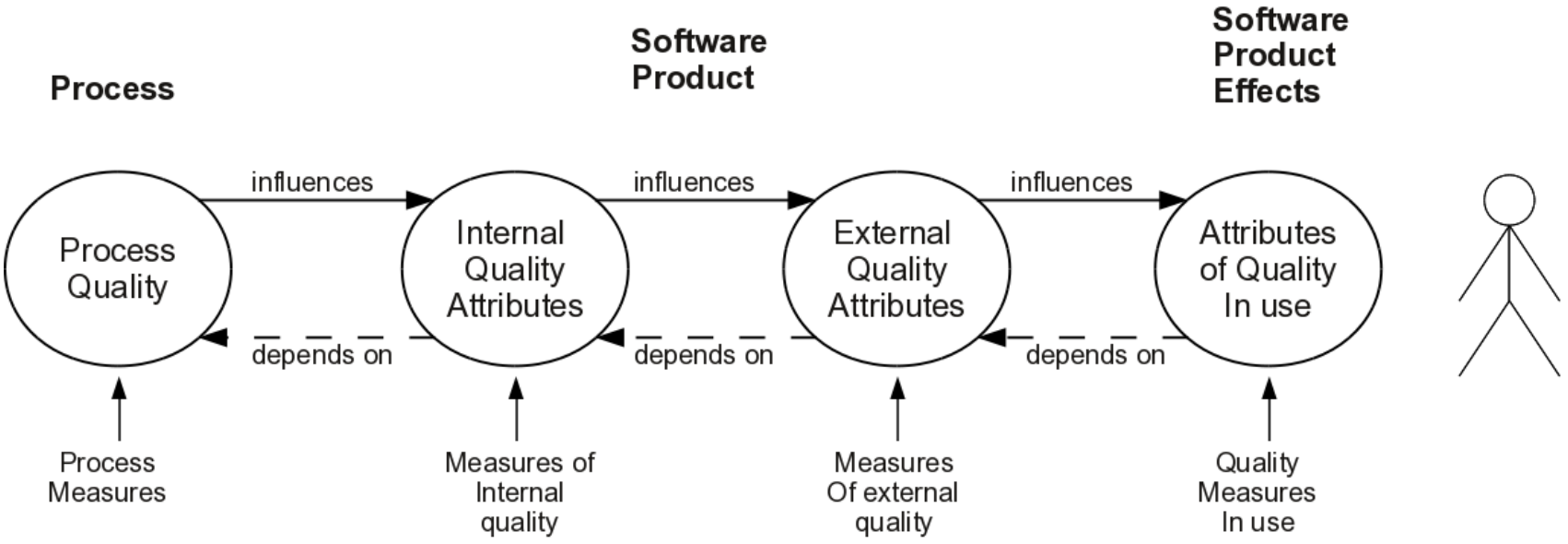
EAF = Effort Adjustment Factor
SF = Scale Factors

Internal vs External Attributes (1/4)

- Generally, 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

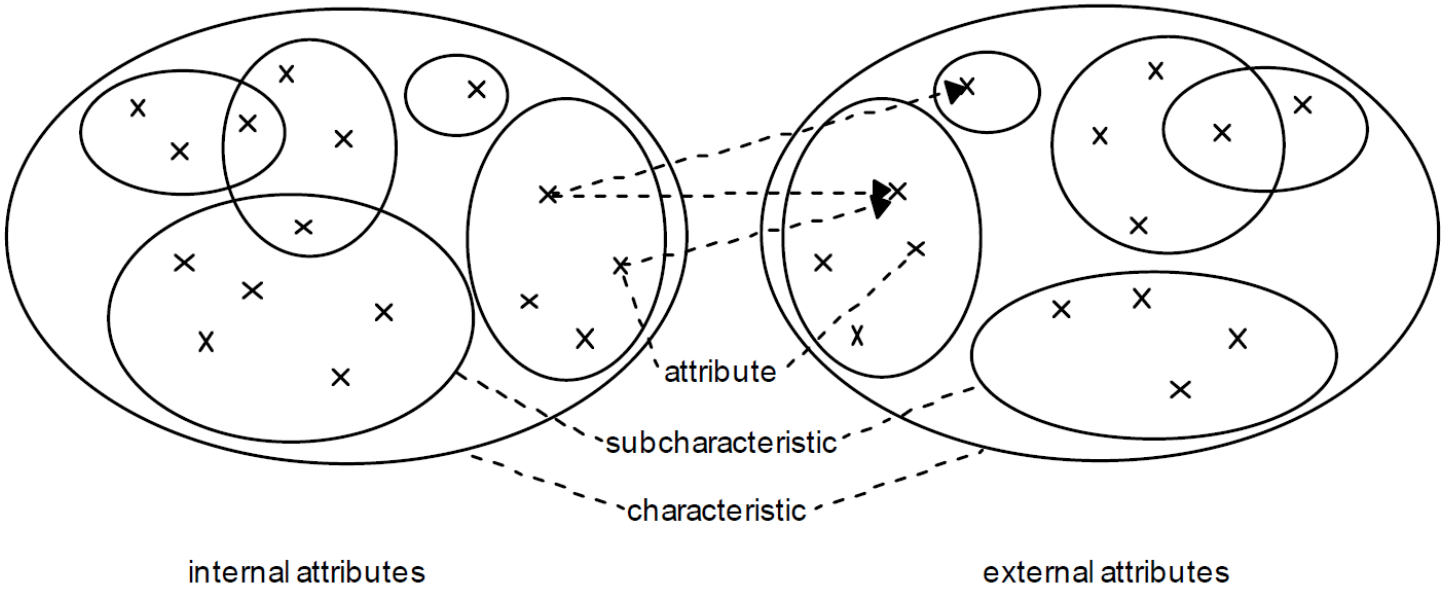
Internal vs External Attributes (2/4)

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



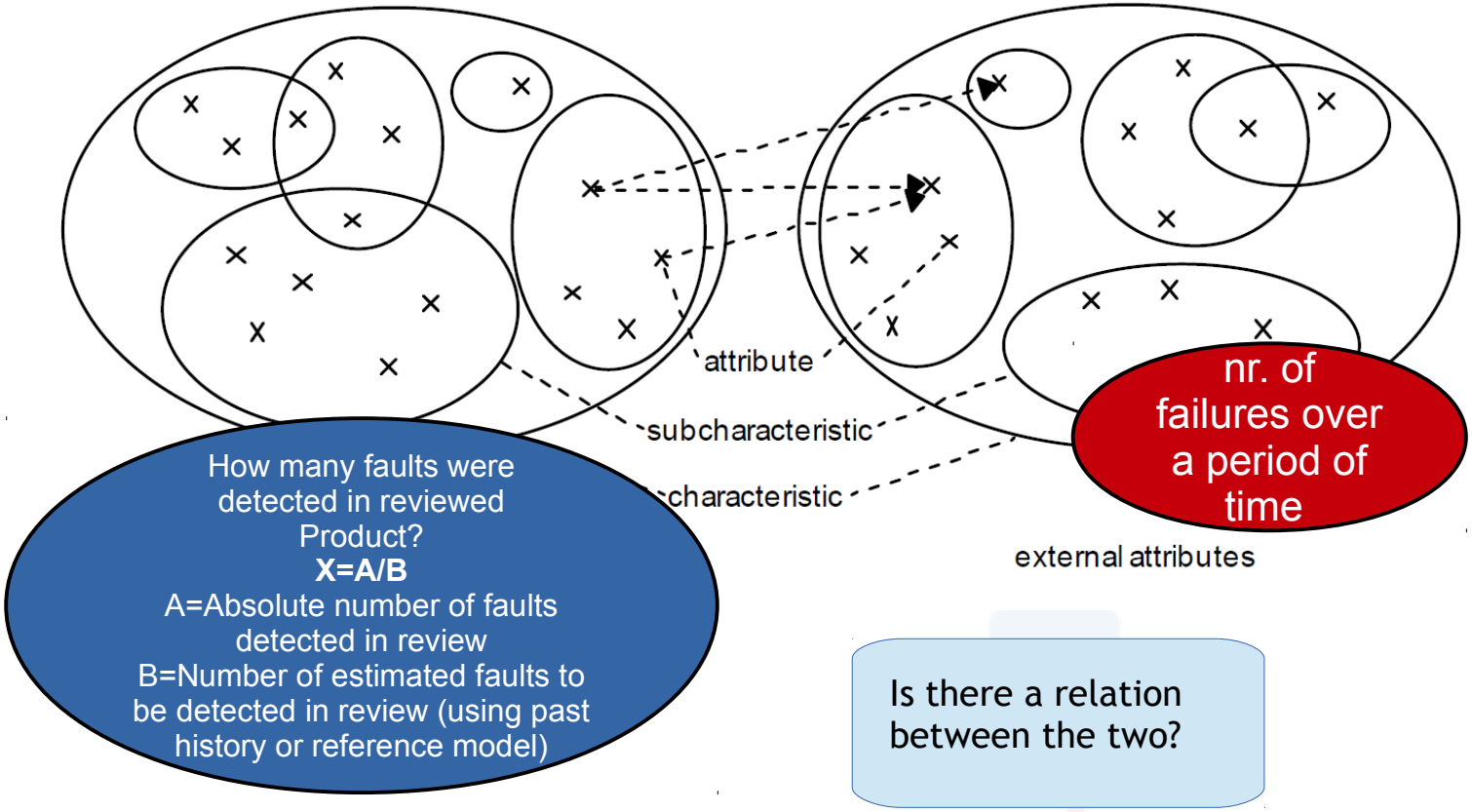
Internal vs External Attributes (3/4)

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



Internal vs External Attributes (4/4)

- The mapping of internal attributes to external ones – and then quality in use – is not as straightforward (example: **reliability**)



ASSUMPTION (!) → fix internal mistakes to fix the corresponding failure(s)

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

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…?

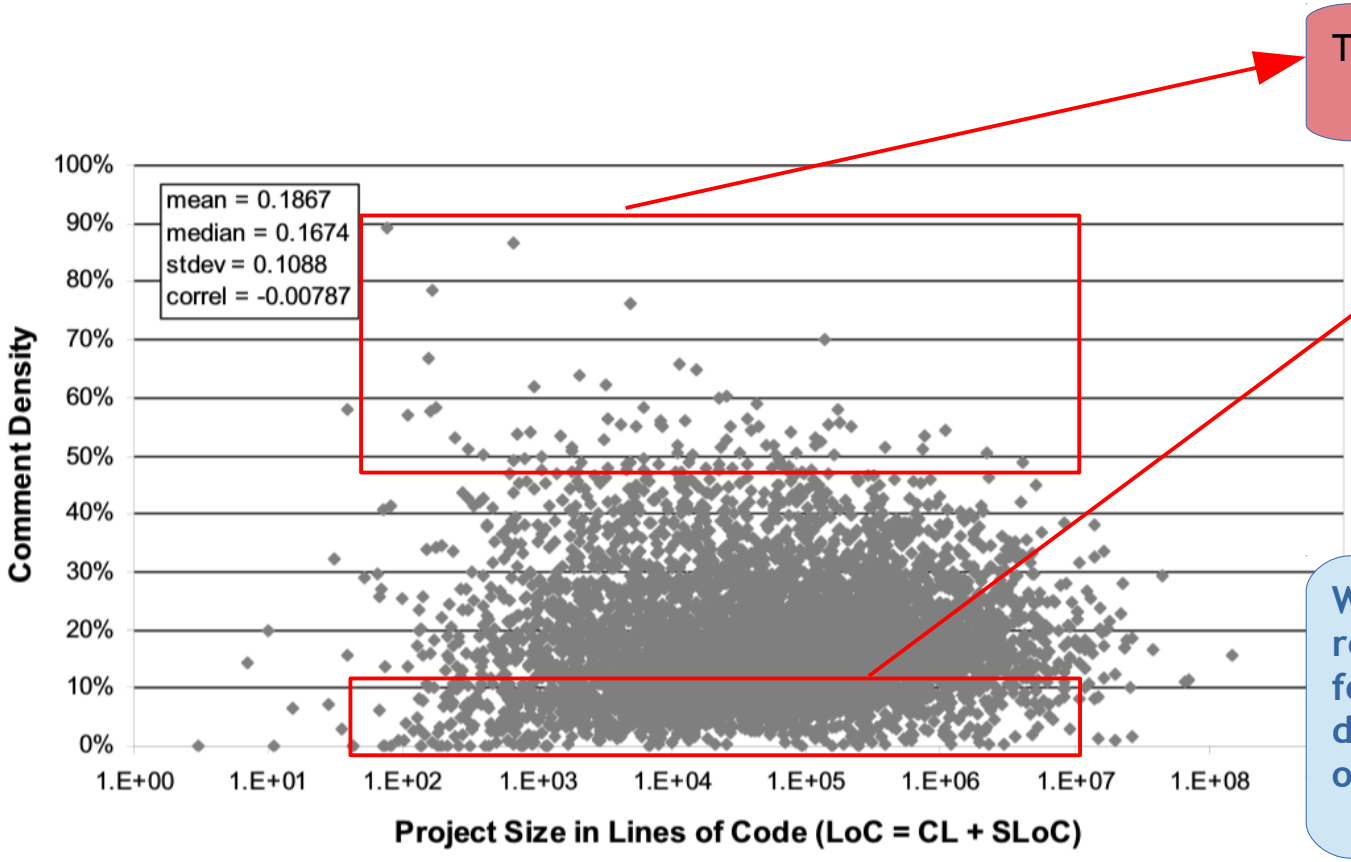
- Size is used for **normalization of existing measures**

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

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

Measures of Size good for…?

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



These look "scary"

What is a good reference value for "comments density" in your opinion?

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...

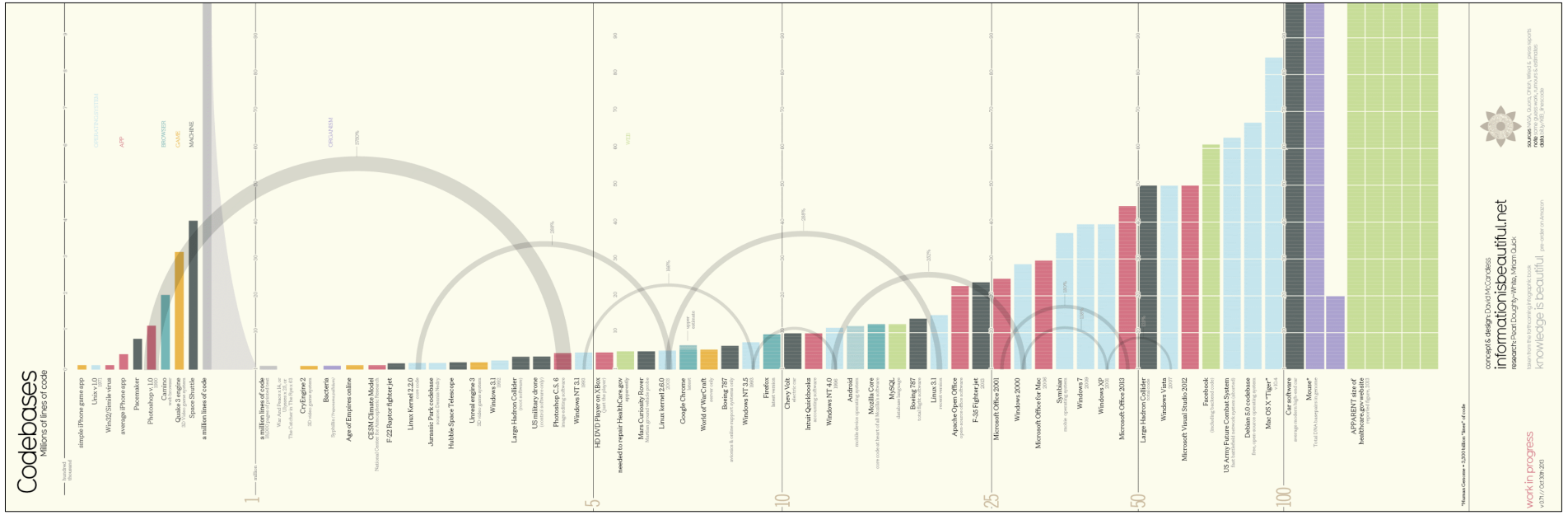
Software	LOCs
Microsoft Windows Vista	~50M
Linux Kernel 3.1	~15M
Android	~12M
Mozilla Firefox	~10M
Unreal Engine 3	~2M

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

→ Measures of source code size should *never* be used to assess the productivity of developers

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/

Another 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

McCabe's Cyclomatic Complexity (CC)

- CC represents the number of independent control flow paths
- G=(N,E) is a graph representing the control flow of a program. N=nodes, E=edges, P = nr. disconnected parts of G, like main program and method call
- Cyclomatic Complexity is defined as:

  v(G) = |E|-|N|+ 2P

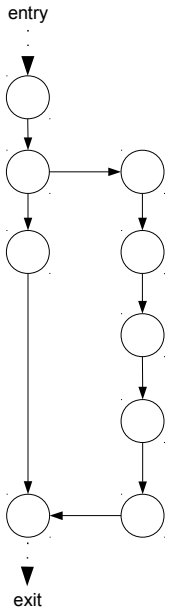
Note: a shortcut is to use # branches + 1 (if, for, foreach, while, do-while, case label, catch, conditional statements)

→ **Assumptions:** higher complexity of the program flow graphs, more complex testing process for the source code

McCabe's Cyclomatic Complexity (CC)

CC = 2

CC of method generatePrimes
v(G)=|E|-|N|+2
v(G)=9-9+2=2

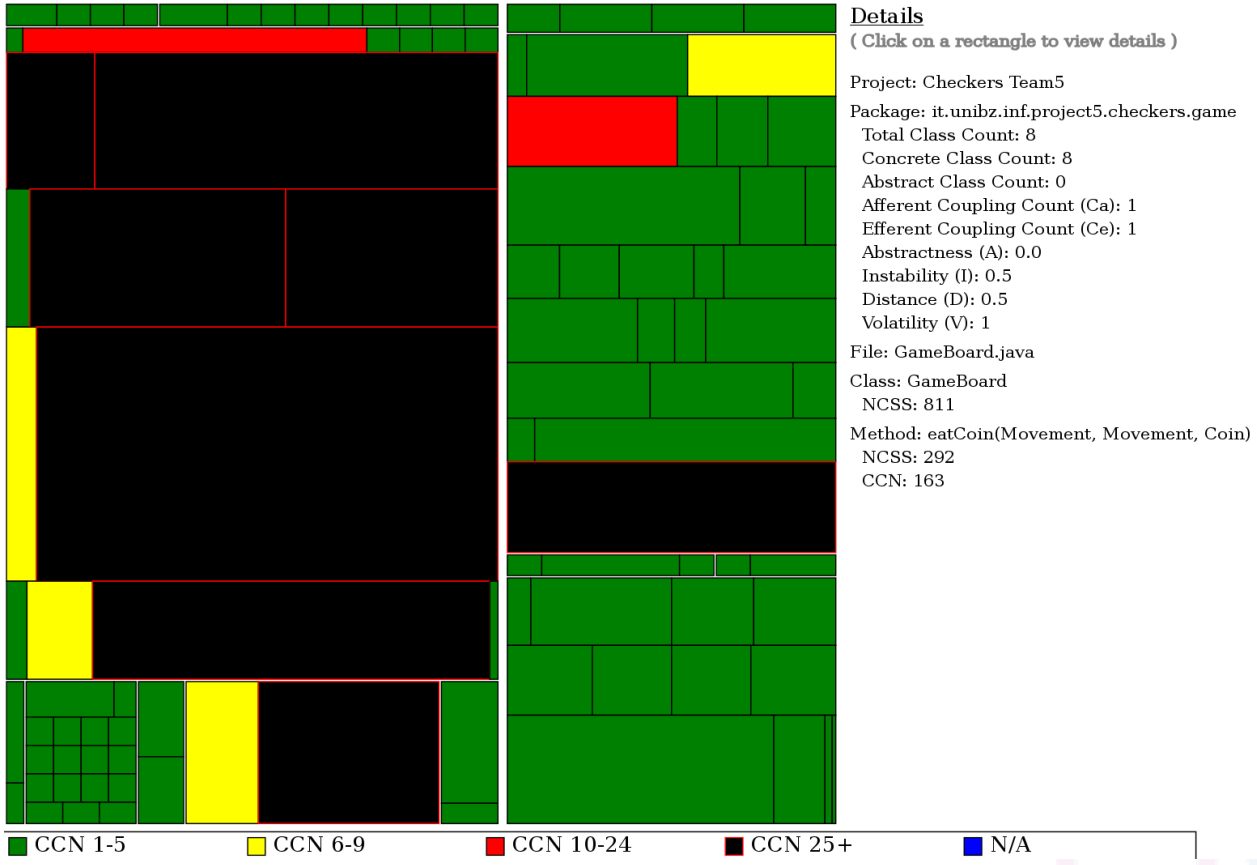


Typical ranges
1-4 low
5-7 medium
8-10 high
11+ very high

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

Example Application of CC

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



Example Application of 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 Application of 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 semantic complexity**
- Both of the following code fragments have the ***same*** Cyclomatic Complexity
  → **which code fragment is easier to understand?**

```
[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

Chidamber & Kemerer Suite

- **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 in Methods

WMC (1/3)

- **WMC**: Weighted methods per class
  - weighted sum of the number of methods of a class. Given C a class and M_1, …, M_K k methods with complexity c_1,…,c_K

$$\sum_{i-1}^{n} c_i, \textit{where c is the complexity of a method}$$

WMC (2/3)

```
class A {
public:
    A() {
        attr = 1;
    }
    void aMethod(){
        attr++;
    }
    void aMethod(int a){
        attr+=a;
    }
    void anotherMethod(){
        B b; C c; D d;
        b.f();
        attr = c.someValue + d.g();
    }
    ~A() { }
private:
    int attr;
};
class B : public A {
public:
    void f(){  }
};
class C: public A {
public:
    int someValue;
};
class D: public C {
public:
    int g() { }
};
class E {
    int a,b,c,d,e,x,y,z;
    void M1(){ a=b=c=d=e; }
    void M2(){ a=b=e; }
    void M3(){ x=y=z; }
};
class F : public C { };
class G : public B, public D { };
```

→ What is the WMC of the following classes?

$$WMC = \sum_{i-1}^{n} c_i$$

WMC (3/3)

```
class A {
public:
    A() {
        attr = 1;
    }
    void aMethod(){
        attr++;
    }
    void aMethod(int a){
        attr+=a;
    }
    void anotherMethod(){
        B b; C c; D d;
        b.f();
        attr = c.someValue + d.g();
    }
    ~A() { }
private:
    int attr;
};
class B : public A {
public:
    void f(){  }
};
class C: public A {
public:
    int someValue;
};
class D: public C {
public:
    int g() { }
};
class E {
    int a,b,c,d,e,x,y,z;
    void M1(){ a=b=c=d=e; }
    void M2(){ a=b=e; }
    void M3(){ x=y=z; }
};
class F : public C { };
class G : public B, public D { };
```

→ What is the WMC of the following classes?

$$WMC = \sum_{i-1}^{n} c_i$$

WMC(A) = NoM(A) = 5
WMC(B) = NoM(B) = 1
WMC(C) = NoM(C) = 0
WMC(D) = NoM(D) = 1
WMC(E) = NoM(E) = 3
WMC(F) = NoM(F) = 0
WMC(G) = NoM(G) = 0

DIT & NOC (1/2)

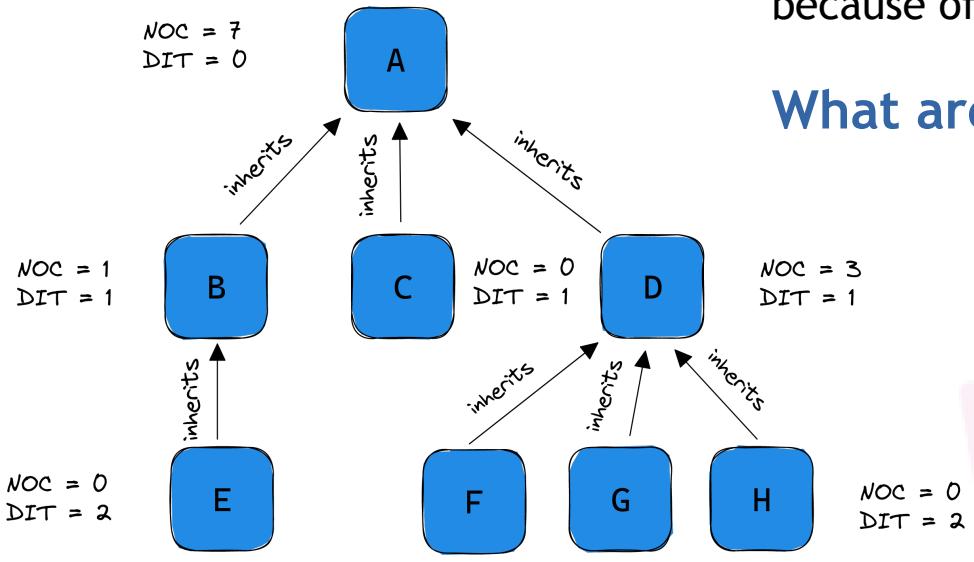
- **DIT:** Depth of Inheritance Tree

  → max inheritance level from the root to the class
- **NOC:** Number of Children

  → nr. Of direct descendants of a class

DIT & NOC (2/2)

- **DIT:** Depth of Inheritance Tree
  → max inheritance level from the root to the class
- **NOC:** Number of Children
  → nr. Of direct descendants of a class



→ The deeper a class is in the hierarchy, the more methods it is likely to inherit, making it more complex
→ Deep trees as such indicate greater design complexity
→ As a positive factor, deep trees promote reuse because of method inheritance

What are "good" DIT & NOC values?

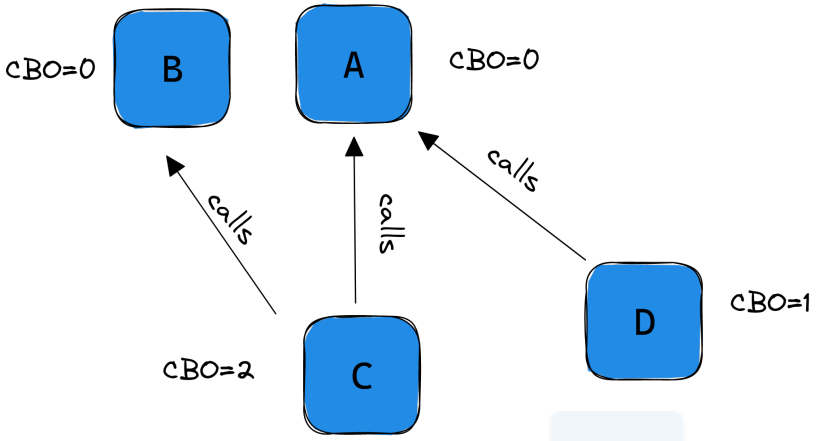
CBO (1/3)

- **CBO:** Coupling between Objects

  → Class A coupled with B, if A is using methods/attributes of B

  → Multiple accesses to the same class are counted as one access

  → High CBO is *undesirable:* Excessive coupling between object classes is detrimental to modular design and prevents reuse



Note: Some definitions of CBO consider both A using B (fan-out), but also B using A (fan-in) for the computation of CBO

CBO (2/3)

```
class A {
public:
    A() {
        attr = 1;
    }
    void aMethod(){
        attr++;
    }
    void aMethod(int a){
        attr+=a;
    }
    void anotherMethod(){
        B b; C c; D d;
        b.f();
        attr = c.someValue + d.g();
    }
    ~A() { }
private:
    int attr;
};
class B : public A {
public:
    void f(){  }
};
class C: public A {
public:
    int someValue;
};
class D: public C {
public:
    int g() { }
};
class E {
    int a,b,c,d,e,x,y,z;
    void M1(){ a=b=c=d=e; }
    void M2(){ a=b=e; }
    void M3(){ x=y=z; }
};
class F : public C { };
class G : public B, public D { };
```

→ What is the CBO of the following classes?

CBO (3/3)

```
class A {
public:
    A() {
        attr = 1;
    }
    void aMethod(){
        attr++;
    }
    void aMethod(int a){
        attr+=a;
    }
    void anotherMethod(){
        B b; C c; D d;
        b.f();
        attr = c.someValue + d.g();
    }
    ~A() { }
private:
    int attr;
};
class B : public A {
public:
    void f(){  }
};
class C: public A {
public:
    int someValue;
};
class D: public C {
public:
    int g() { }
};
class E {
    int a,b,c,d,e,x,y,z;
    void M1(){ a=b=c=d=e; }
    void M2(){ a=b=e; }
    void M3(){ x=y=z; }
};
class F : public C { };
class G : public B, public D { };
```

→ What is the CBO of the following classes?

CBO(A)=3

CBO(B)=CBO(C)=CBO(D)=

CBO(E)=CBO(F)=0

RFC (1/3)

- **RFC:** Response for a Class $RFC = |M_c \cup M_e|$

  → the number of methods of a class than can be invoked in response of a call to a method of a class

  - count of methods that can be executed by class A responding to a message (M_c)
  - Sum all to external calls to other methods (only count one call to the same method once)

  → A large RFC has been found to indicate more faults

  → Classes with a high RFC are more complex and harder to understand: Testing and Debugging are more complicated

RFC (2/3)

```
class A {
public:
    A() {
        attr = 1;
    }
    void aMethod(){
        attr++;
    }
    void aMethod(int a){
        attr+=a;
    }
    void anotherMethod(){
        B b; C c; D d;
        b.f();
        attr = c.someValue + d.g();
    }
    ~A() { }
private:
    int attr;
};
class B : public A {
public:
    void f(){  }
};
class C: public A {
public:
    int someValue;
};
class D: public C {
public:
    int g() { }
};
class E {
    int a,b,c,d,e,x,y,z;
    void M1(){ a=b=c=d=e; }
    void M2(){ a=b=e; }
    void M3(){ x=y=z; }
};
class F : public C { };
class G : public B, public D { };
```

→ What is the RFC of the following classes?

$$RFC = \left| M_c \cup M_e \right|$$

RFC (3/3)

```
class A {
public:
    A() {
        attr = 1;
    }
    void aMethod(){
        attr++;
    }
    void aMethod(int a){
        attr+=a;
    }
    void anotherMethod(){
        B b; C c; D d;
        b.f();
        attr = c.someValue + d.g();
    }
    ~A() { }
private:
    int attr;
};
class B : public A {
public:
    void f(){  }
};
class C: public A {
public:
    int someValue;
};
class D: public C {
public:
    int g() { }
};
class E {
    int a,b,c,d,e,x,y,z;
    void M1(){ a=b=c=d=e; }
    void M2(){ a=b=e; }
    void M3(){ x=y=z; }
};
class F : public C { };
class G : public B, public D { };
```

→ What is the RFC of the following classes?

$$RFC = \left| M_c \cup M_e \right|$$

```
RFC(A) = 7
WMC(B) = NoM(B) = 1
WMC(C) = NoM(C) = 0
WMC(D) = NoM(D) = 1
WMC(E) = NoM(E) = 3
WMC(F) = NoM(F) = 0
WMC(G) = NoM(G) = 0
```

LCOM (1/2)

- **LCOM:** Lack of cohesion in methods
  - How closely the local methods are related to the local instance variables in the class
  - We use a "negative" measure of cohesiveness, the lack of cohesion of its methods

Take each field in the class, count the methods that reference it, sum all together for all fields.

$$LCOM = 1 - \frac{\sum F\,|M_f|}{|M| \; x \; |F|}$$

Divide by the # of methods multiplied the # of fields

M = static and instance methods in the class
F = instance field in the class
M_f = methods accessing field f
|S| = cardinality of set S

LCOM (2/2)

- **LCOM:** Lack of cohesion in methods

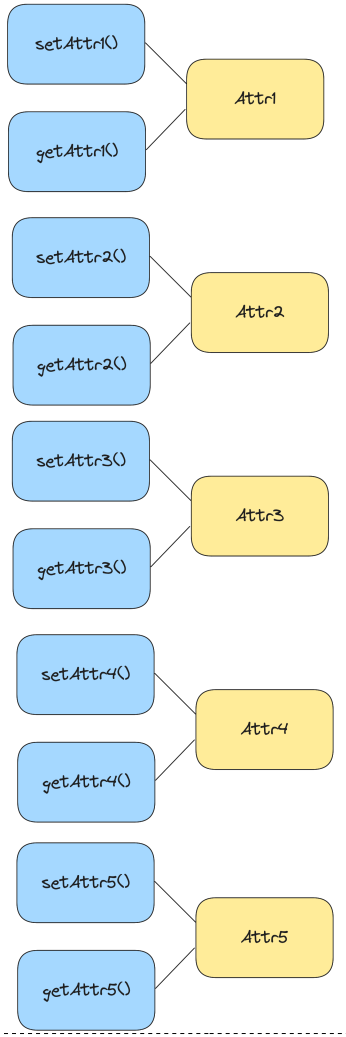
$$LCOM = 1 - \frac{\sum F |M_f|}{|M| x |F|}$$

Take each field in the class, count the methods that reference it, sum all together for all fields.

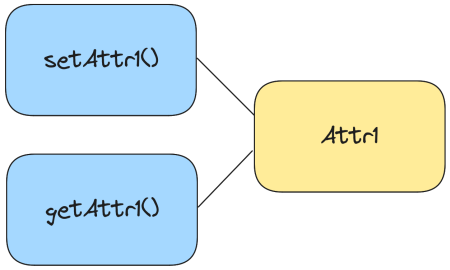
Divide by the # of methods multiplied the # of fields

M = static and instance methods in the class
F = instance field in the class
M_f = methods accessing field f
|S| = cardinality of set S

Class A $LCOM = 1 - \frac{10}{50} = 0.8$



Class B $LCOM = 1 - \frac{2}{2} = 0$



Examples inspired from NDepend documentation

Question Time

Maintainability Index (MI)

- Given all that we have seen, what are your thoughts about the following metric computing the Maintainability Index (MI) of a project:

$$MI = 171 - 5.2 \cdot \ln(V) - 0.23 \cdot CC - 16.2 \cdot \ln(LOC)$$

CC = Cyclomatic Complexity as defined previously

LOC = Lines of Code

V is the **Halstead volume**, measuring the complexity of code based on length and vocabulary used (in the code)

$$V = N * \log_2 n$$

$$where\ N = N_1 + N_{2,}$$

$$N_1 = Total\ operators(like >, ;,), etc.., N_2 = Total\ operands(like\ j, i, 0, etc...)$$

$$N = n_1 + n_{2,}$$

$$n_1 = unique\ operators, n_2 = unique\ operands$$

In your view, what is good and what is bad about this metric?

Note: you might see different versions of MI implemented in different tools – this is the original formula that has a range (171,-∞), other variations go in the (0,100) range, e.g. look at Microsoft Visual Studio documentation for details

The Goal Question Metrics (GQM) Approach

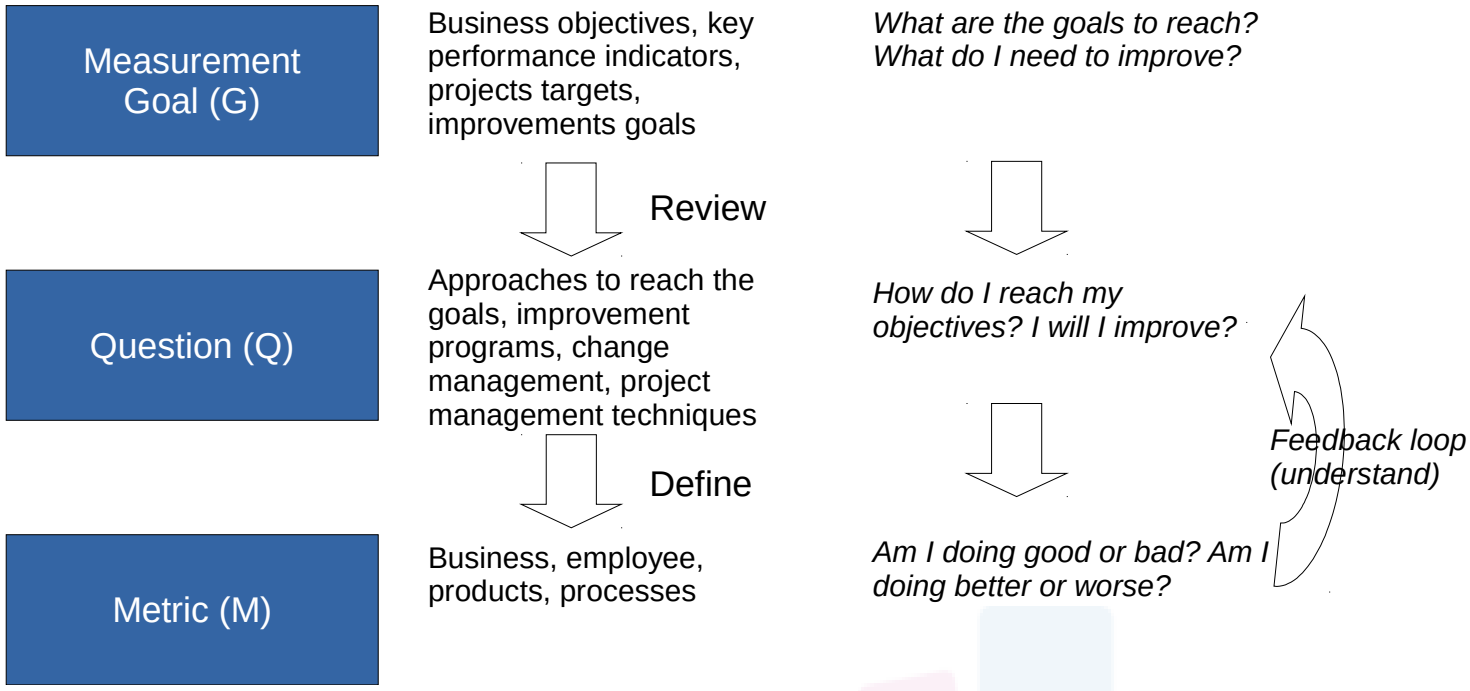
[Briefly]

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)
- 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:



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...

  → **Point of view**: manager, developer, tester, customer, etc...

  → **Perspective or Focus**: cost, effectiveness, correctness, defects, changes, product measures, etc...

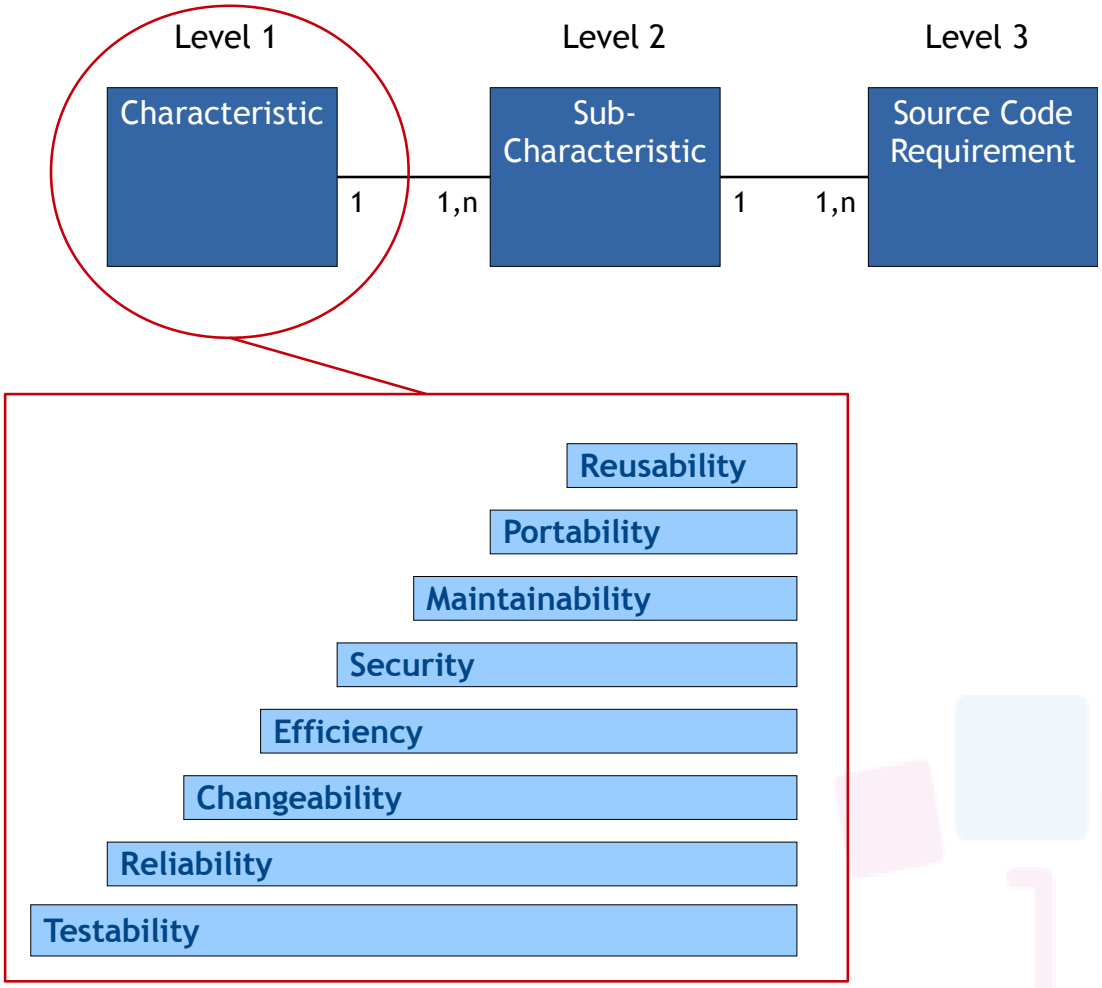
  → **Environment or Context**: specify the environmental factors, including process factors, people factors, problem factors, methods, tools, constraints, etc...

SQALE (Software Quality Assessment Based on Lifecycle Expectations)

[Briefly]

SQALE

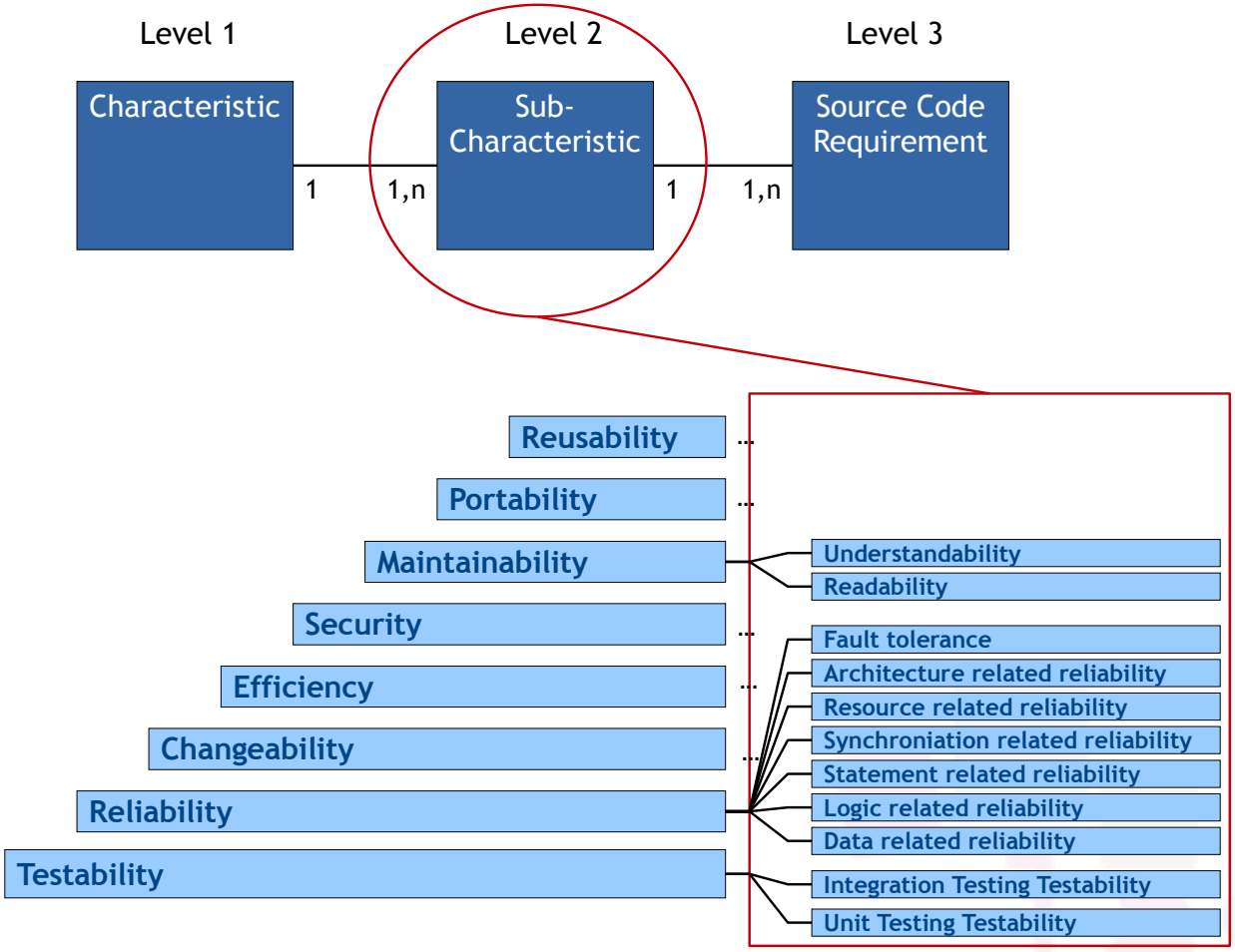
- SQALE (Software Quality Assessment Based on Lifecycle Expectations) is a quality method to **evaluate technical debts** in software projects **based on the measurement of software characteristics**
  - Three levels, the first one including 8 software characteristics



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

SQALE

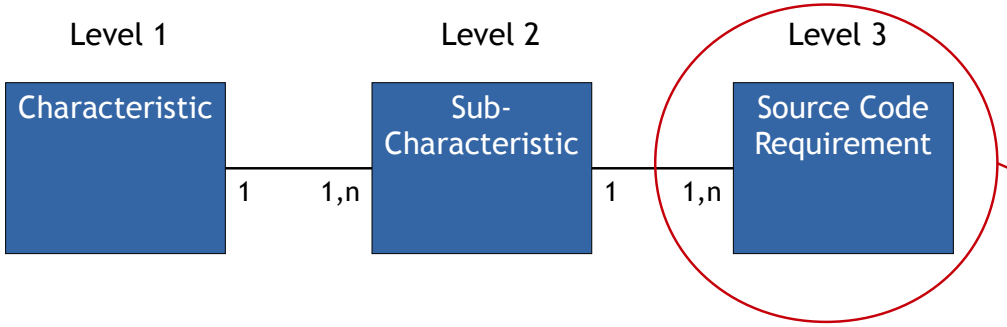
- The second level is formed by characteristics

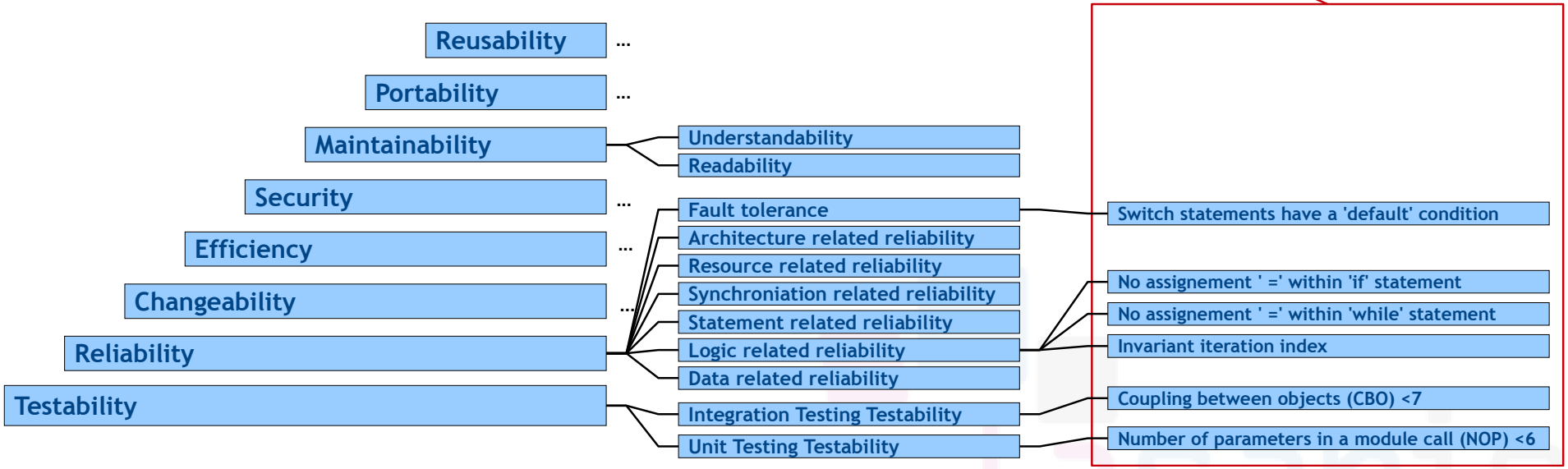


Adapted from: http://www.sqale.org/wp-content/uploads/2010/08/SQALE-Method-EN-V1-0.pdf

SQALE

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





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

Remediation & non-Remediations

SQALE – Remediation Function

- For each of the source code requirements we need to associate a **remediation function** that translates the non-compliance 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 functions 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 - 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 artifact that has an estimated development cost of 300 hours and a SQALE Testability Index (STI) of 8.30 hours, using the reference table on the left

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

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

Closing Remarks

Conclusions - 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

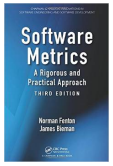
Count what is countable.
Measure what is measurable.
And what is not measurable, make measurable.
Galileo Galilei

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

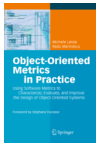
References

- 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