Lean Software Development
Existing Software Measures
Lectures 5 and 6

Artem Kruglov, Giancarlo Succi

Innopolis University

a.kruglov@innopolis.ru, *g.succi@innopolis.ru*

2nd and 3rd February 2021

Initial Classification for Metrics

- Size Oriented Metrics
- Function Oriented Metrics
- Object-Oriented Metrics
- Chidamber and Kemerer (CK) Suite

Typical Size-Oriented Metrics

- Errors per KLOC (thousand lines of code)
- Defects per KLOC
- $ per LOC
- Page of documentation per KLOC
- Errors / person-month
- LOC per person-month
- $ / page of documentation

Typical Function-Oriented Metrics

- Errors per FP (thousand lines of code)
- Defects per FP
- $ per FP
- Pages of documentation per FP
- FP per person-month

Size Oriented Metrics

Lines of Code

- Need for a standard (a normalization)
  - For instance we use the count of the ";"
- Once a standard is set they can be computed automatically (Objective metrics)
- ***They MUST not be used to evaluate people productivity (easy to alter!!!)***
- When used properly, they work!!

Example (with our definition)

```
void f() {
    while (!done){
        count++; //1
       if(count > 10){
            fixed_count = fixed_count + count; //2
            done = 1; //3
        } else if(count >5){
            fixed_count --; //4
        } else {
            fixed_count = count * 4; //5
        }
    } // while
}
```

$$LOC = 5!$$

Example (with another definition)

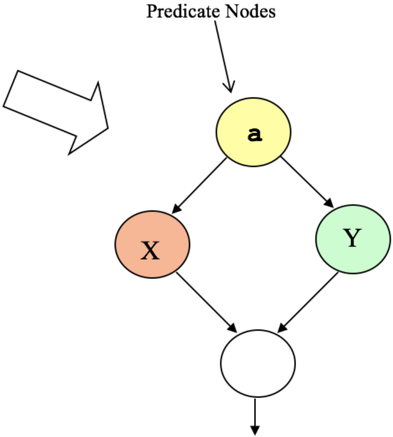
```
void f() { //1
    while (!done){ //2
        count++; //3
       if(count > 10){
            fixed_count = fixed_count + count; //4
            done = 1; //5
        } else if(count >5){ //6
            fixed_count --; //7
        } else { //8
            fixed_count = count * 4; //9
        } //10
    } //11
} // 12
```

$LOC = 12!$

The quest for productivity measures

- In how many ways I can write a sequence of 10 **sqrt()s**???
- Usually productivity is defined "Output/Input"
- But what is the Output in SW? What is the Input?
- LOC/effort is evaluating the volume throughput, taking into account the input volume!

```
if (a) {
        X();
} else {
        Y();
}
```

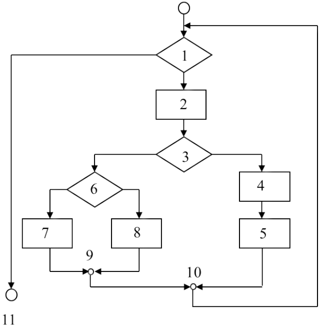


from Pressman pg 459

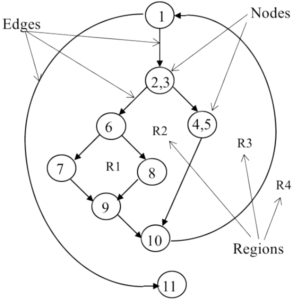
Introducing Cyclomatic Complexity

```
void f() {
    while (!done){
        count++;
       if(count > 10){
            fixed_count = fixed_count + count;
            done = 1;
        } else if(count >5){
            fixed_count --;
        } else {
            fixed_count = count * 4;
        }
    } // while
}
```

Flow Graph Notation



Flow Chart



Flow Graph

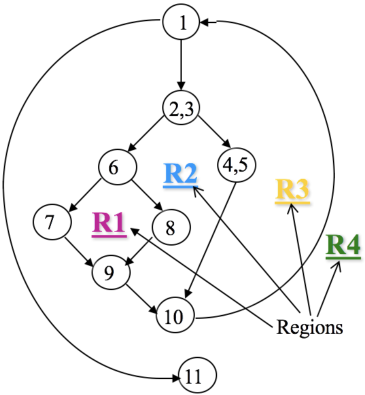
Flow Chart

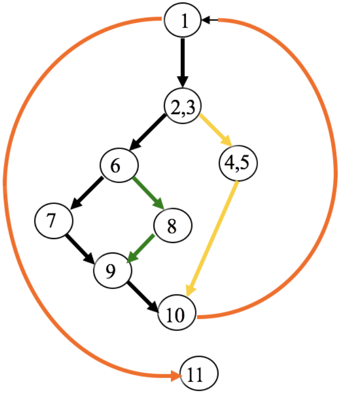
Flow Graph

Cyclomatic Complexity (Def)

V(G) = #Regions in the Graph
V(G) = #Independent Paths in the Graph
V(G) = E - N + 2, *where*
E = number of edges and
N = number of nodes
V(G) = P + 1, *where*
P = number of predicated nodes (i.e., if, case, while, for, ...)

Computing CC (definitions)

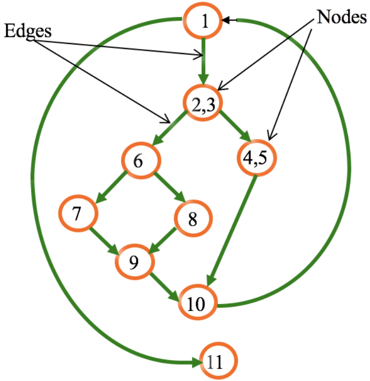


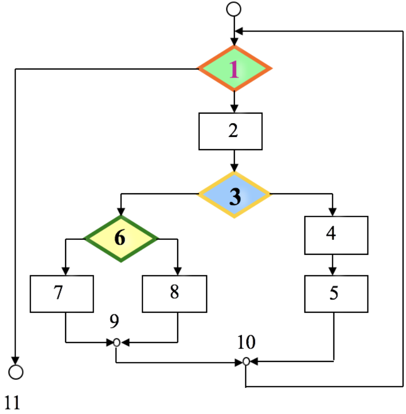


4 regions!!!

4 independent paths!!!

Computing CC (formula)





11 Edges, 9 Nodes . . .
$11 - 9 + 2 = 4$!!!

$3 + 1 = 4$!!

Fan In and Fan Out

- The Fan In of a module is the amount of information that "enters" the module
- The Fan Out of a module is the amount of information that "exits" a module
- We assume all the pieces of information with the same size
- Fan In and Fan Out can be computed for functions, modules, objects, and also non-code components

Computing Fan In and Fan Out

Usually:

- Parameters passed by values count toward Fan In
- External variables used before being modifies count toward Fan In
- External variables modified in the block count toward Fan Out
- Return values count toward Fan Out
- Parameters passed by reference ... depend on their use...

```c
#define<stdio.h>
#define<math.h>

int globalInVar = 9;
int globalOutVar;

float Simple(float x, float y){
      int a;
      float z;
      z = sqrt( x + y +
            globalInVar);
      globalOutVar = int(z+2);
      return z;
}
```

Code line	fan-in	fan-out
float Simple(float x, float y){	2	
globalInVar);	1	
globalOutVar = int(z+2);		1
return z;		1
	3	2

Simple Example of Fan In / Fan Out

```
#define<stdio.h>
#define<math.h>

int globalInVar = 9;
int globalOutVar;

float Simple(float x, float y){
   int a;
   float z;
   z = sqrt( x + y +
             globalInVar);
   globalOutVar = int(z+2);
   return z;
}
```

Proposed Exercises

#define<stdio.h>		
#define<math.h>	**fan-in**	**fan-out**
int globalVarA = 0;		
int globalVarB = 3;		
float global VarC = 7.0;		
float chechValue(float x, float y){	**2**	
int a;		
float z;		
z = sqrt(x + y + globalVarC);	**1**	
globalVarA ++;	**1**	**1**
a = globalVarB;	**1**	
globalVarC = z + (float)globalVarA;		**1**
return z;		**1**
}		
Total	**5**	**3**

Proposed Exercises

Please refer to the handout

Halstead Software Science

Primitive Measures	n_1– number of distinct operators
	n_2– number of distinct operands
	N_1– total number of operator occurrences
	N_2– total number of operand occurrences

Types:	$n = n_1 + n_2$
Tokens / Length:	$N = N_1 + N_2$
Volume:	$V = Nlog_2(n)$
Volume Ratio:	$L = (2/n_1) \times (n_2/N_2)$
Program Level:	$PL = \frac{1}{(n_1/2)\times(N_2/n_2)}$
Software Science Effort:	$E = V/PL$
Time for testing:	$T = E/18$ in seconds.

Halstead's Metrics (example)

```
#define<stdio.h>
#define<math.h>

int globalVarA = 0;
int globalVarB = 3;
float global VarC = 7.0;

float chechValue( float x, float y){
        int a;
        float z;
        z = sqrt( x + y + globalVarC );
        globalVarA ++;
        a = globalVarB;
        globalVarC = z + (float)globalVarA;
        return z;
}
```

Operators:	**Count:**
()	3
{}	1
=	6
+	5
int	3
float	6
sqrt	1
checkValue	1
return	1
$n_1 = 9$	$N_1 = 27$

N = 44; n = 16; V = 176; PL =0.09; E = 1923.42' = 32'

Halstead's Metrics (example cont.)

```
#define<stdio.h>
#define<math.h>

int globalVarA = 0;
int globalVarB = 3;
float global VarC = 7.0;

float chechValue( float x, float y){
        int a;
        float z;
        z = sqrt( x + y + globalVarC );
        globalVarA ++;
        a = globalVarB;
        globalVarC = z + (float)globalVarA;
        return z;
}
```

Operands:	**Count:**
globalVarA	3
globalVarB	2
globalVarC	3
x	2
y	2
a	2
z	3
$n_2 = 7$	$N_2 = 17$

N = 44; n = 16; V = 176; PL =0.09; E = 1923.42 = 32'

Function Oriented Metrics

Function Points

- Function Points are a measure of *how big* is the program, independently from the actual physical size of it
- It is a weighted count of several features of the program
- Dislikers claim FP make no sense wrt the representational theory of measurement
- There are firms and institutions taking them very seriously

Analyze information   **domain of the**   Establish count for input domain and   **application**   system interfaces   **and develop counts**
$\Downarrow$
Weight each count by   Assign level of complexity or weight   **assessing complexity**   to each count
$\Downarrow$
Assess influence of   Grade significance of external factors, F_i

Analyzing the Information Domain

Measurement parameter *(number of)*	**count**		**Weighting Factor** **simple**	**average**	**complex**		
inputs	☐	x	3	4	6	=	☐
user outputs	☐	x	4	5	7	=	☐
user inquiries	☐	x	3	4	6	=	☐
files	☐	x	7	10	15	=	☐
ext. interfaces	☐	x	5	7	10	=	☐

Assuming all inputs with the same weight, all output with the same weight, ...

Complete Formula for the Unadjusted Function Points:

$$\sum_{Inputs} W_i + \sum_{Outputs} W_o + \sum_{Inquiry} W_{in} + \sum_{IntFiles} W_{if} + \sum_{ExtInt} W_{ei}$$

Taking Complexity into Account

Factors are rated on a scale of 0 (not important) to 5 (very important):

data communications	on-line update
distributed functions	complex processing
heavily used configuration	installation ease
translation rate	operation ease
on-line data entry	multiple sites
end user efficiency	facilitate change

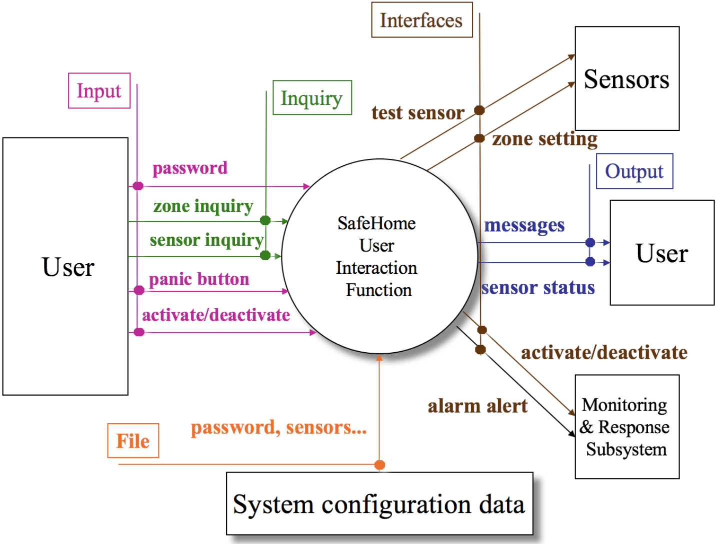
Formula:

$$CM = \sum_{ComplexityMultiplier} F_{ComplexityMultiplier}$$

Measurement parameter *(number of)*	**count**		**Weighting Factor** simple	average	complex		
inputs	☐	x	3	4	6	=	☐
user outputs	☐	x	4	5	7	=	☐
user inquiries	☐	x	3	4	6	=	☐
files	☐	x	7	10	15	=	☐
ext. interfaces	☐	x	5	7	10	=	☐
Unadjusted Function Points:							
count total						=	☐
complexity mpl.						=	☐
function points						=	☐

$$FP = UFP \times (0.65 + 0.01 \times CM)$$

Safe Home Example



Safe Home Example

Measurement parameter *(number of)*	**count**		**Weighting Factor** simple	average	complex		
inputs	$\boxed{3}$	x	3	4	6	=	$\boxed{9}$
user outputs	$\boxed{2}$	x	4	5	7	=	$\boxed{8}$
user inquiries	$\boxed{2}$	x	3	4	6	=	$\boxed{6}$
files	$\boxed{1}$	x	7	10	15	=	$\boxed{7}$
ext. interfaces	$\boxed{4}$	x	5	7	10	=	$\boxed{20}$
count total							$\boxed{\mathbf{50}}$

Using $FP = \text{count total} \times (0.65 + 0.01 \times \sum F_i)$

where $\sum F_i = 46$, we get

$$FP = 50 \times (0.65 + 0.01 \times 46)$$

$$FP = 56$$

New stuff on function points

- Feature points
  - To deal also with engineering and real time applications
  - Not much used, however ...
- Software Backfiring
  - to link function points to LOC on the average, since there are (poor) techniques to estimate the duration of a software project, given the predicted lines of code (COCOMO2, Putnam model)

Quality Metrics

- A means to specify the features of a software product and to verify them
- ISO 9126 identifies 6 characteristics and 21 sub-characteristics
- Each characteristic is defined by a ***disjoint subset*** of sub-characteristics that are then used to its evaluation

ISO 9126 Characteristics

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Other Kinds of Metrics

Other Kinds of Metrics

There are MANY other kinds of metrics:

- Metrics for requirements
- Metrics for analysis
- Metrics for design
- Metrics for testing
- Object oriented metrics
- ...

The CK Metrics suite

- The CK metrics suite is a set of metrics for object oriented systems
- It tries to capture different kind of properties of OO
- It attempts to compromise between providing a careful description and a the principle of parsimony

Metrics in the CK suite

- Weighted Methods per Class (WMC)
- Depth of Inheritance Tree (DIT)
- Number of Children (NoC)
- Coupling between Objects (CBO)
- Response for a Class (RFC)
- Lack of COhesion in Methods (LCOM)

WMC

Let C be a class with methods $M_1, M_2, \ldots M_n$
with complexity $c_1, c_2, \ldots c_n$
$WMC(C) = \sum c_i$
If all methods are considered to be of complexity one:
$WMC(C) = NoM(C)$

DIT

- The Depth of the Inheritance Tree of a class is the longest path in the inheritance hierarchy from the class to its most remote ancestor
- In case of multiple inheritance, DIT is the largest number among the different most remote ancestors

NoC

- The Number of Children of a class is the number of immediate subclasses of that class

CBO

- Coupling between objects of a class C is the number of other classes to which C is related through association, aggregation, or composition

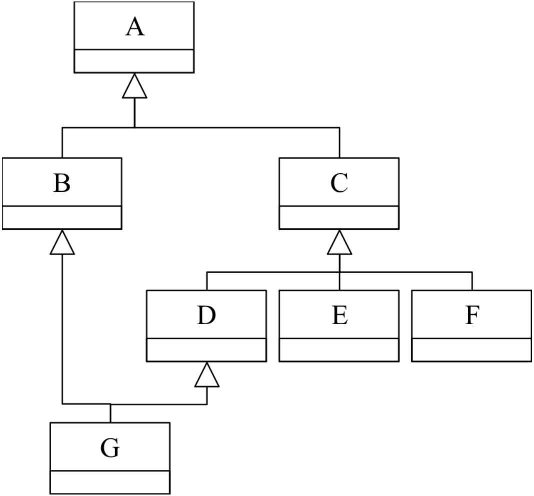
RFC

- Response For a Class is the number of methods (of that class or of other classes) that can be invoked from within the class in response to a call to a method of the class

LCOM

- Let C be a class with methods $M_1, M_2, \ldots M_n$ using each using set of attributes $UA_1, UA_2, \ldots UA_n$
- Consider the sets of Non Cohesive Methods and of Cohesive Methods of C:
  - $NCM(C) = \{< M_i, M_j >: UM_i \cap UM_j = \varnothing, i < j\}$
  - $C(C) = \{< M_i, M_j >: UM_i \cap UM_j \neq \varnothing, i < j\}$
- LCOM(C) = $\mid NCM(C) \mid - \mid C(C) \mid$

Example (1/4)



Example (2/4)

```
class A {
public: A(){}
   void aMethod(){}
   void aMethod(int i){}
   void anotherMethod();
   ~A(){}
private: int attr;
};

class B : public A{ public: void f(){}};

class C : public A{ public: int aValue;};

class D : public C{ public: int g(){}};
```

Example (3/4)

```
class E {
private: int a, b, c, d, e, x, y, z;
public:
  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 {};

void A::anotherMethod() {
  B b; C c; D d; b.f();
  attr = c.someValue + d.g();
}
```

Example (4/4)

```
WMC(A)= |{A(), aMethod(), aMethod(int), anotherMethod(), ~A()}|=5

DIT(G)= max{|{B,A}|,|{D,C,A}|,|{E,C,A}|,|{F,C,A}| = 3

NOC(C)= |{D,E,F}| = 3

CBO(A)= |{B, C, D}| = 3

RFC(A)= |{A(),aMethod(),aMethod(int), anotherMethod(),~A()}| +
    |{B::f(),D::g()}| = 7

LCOM(E)= | {<M1,M3>,<M2,M3>}| - | { <M1,M2> } | = 2 - 1 = 1
```