Introduction to software metics

Alexander Voigt

Technische Universität Dresden
Institut für Kern- und Teilchenphysik

IKTP Computing Kaffee
12 December 2011

Content

Why software metrics?

software metric

software metric = a measure of a property of a piece of software

Goal: objective, reproducible and quantifiable measurements of software for

- schedule planning
- cost estimation
- quality testing
- software debugging
- software performance optimization

Simple/ naive metrics

- Number of lines of code (SLOC)
- Number of lines of comments (NLOCom)
- Number of lines of comments per SLOC
- Number of classes and interfaces
- Program execution time
- Program binary size
- Bugs per line of code

Number of lines of code

free SLOC counter (GNU GPL): `sloccount`

```
~/tmva/ $ sloccount src/*.cxx inc/TMVA/*.h
Total Physical Source Lines of Code (SLOC)                =
  ↪ 48,883
Development Effort Estimate, Person-Years (Person-Months) =
  ↪ 11.88 (142.50)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         =
  ↪ 1.37 (16.46)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  =
  ↪ 8.66
Total Estimated Cost to Develop                           =
  ↪ $ 1,604,196
 (average salary = $56,286/year, overhead = 2.40).
```

Procedural software metrics — McCabe's cyclomatic complexity

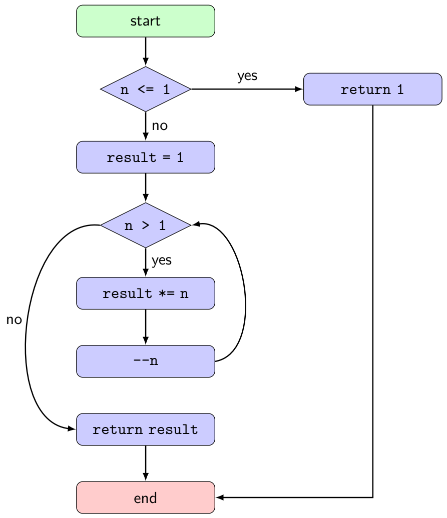
McCabe's Cyclomatic Complexity (MVG) [McC76] = the number of linearly independent paths through the source code

Example:

```
unsigned int factorial(unsigned int n)
{
   if (n <= 1) return 1;
   unsigned int result = 1;
   while (n > 1) {
      result *= n;
      --n;
   }
   return result;
}
```

In this example MVG = 3, see next slide.

Procedural software metrics — McCabe's cyclomatic complexity



McCabe's cyclomatic complexity M is defined as

$$M := E - N + 2P$$

where

N = number of nodes,

E = number of edges,

P = number of connected components

in the corresponding control flow graph.

In this example: $E = 10$, $N = 9$, $P = 1 \Rightarrow M = 3$

Procedural software metrics — Information flow complexity

Information flow complexity (IFC) (defined by IEEE 982.2):

$$\text{IFC} := (\text{fanin} \cdot \text{fanout})^2$$

- fanin = local flows into a procedure + number of data structures from which the procedure retrieves data
- fanout = local flows from a procedure + number of data structures that the procedure updates

Example:

```
int main()
{
   unsigned number = 5;
   cout << number << "! = "        // fanout = 1
        << factorial(number)       // fanin = 1
        << endl;
   return 0;
}
```

$\text{IFC} = (1 \cdot 1)^2 = 1$

Object oriented software metrics

Weighted methods per class (WMC)

Weighted methods per class (WMC)
Variants:

- WMC1: count 1 for each function
- WMCv: count 1 for functions accessible to other modules, 0 for private functions

Example:

```
class A {
public:
   method1() {}
   method2() {}
private:
   method3() {}
   method4() {}
};
```

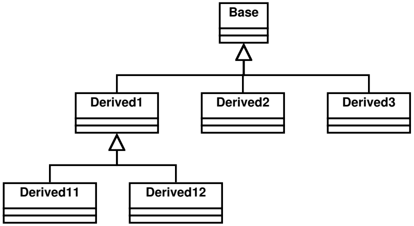
WMC1 = 4
WMCv = 2

Object oriented software metrics

Depth of inheritance tree (DIT)

Depth of inheritance tree (DIT) = The length of the longest path of inheritance ending at the current module.

The deeper the inheritance tree for a module, the harder it may be to predict its behaviour. On the other hand, increasing depth gives the potential of greater reuse by the current module of behaviour defined for ancestor classes.



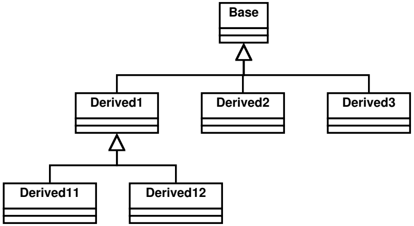
DIT(Base) = 0, DIT(Derived1) = 1, DIT(Derived11) = 2

Object oriented software metrics

Number of children (NOC)

Number of children (NOC) = The number of modules which inherit directly from the current module.

Moderate values of this measure indicate scope for reuse, however high values may indicate an inappropriate abstraction in the design.



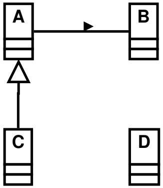
NOC(Base) = 3
NOC(Derived1) = 2
NOC(Derived2) = 0

Object oriented software metrics

Coupling between objects (CBO)

Coupling between objects (CBO) = The number of other modules which are coupled to the current module either as a client or a supplier.

Excessive coupling indicates weakness of module encapsulation and may inhibit reuse.



CBO(A) = 2
CBO(B) = 1
CBO(C) = 1
CBO(D) = 0

Software package metrics — motivation

some OO design goals:

- extend/ alter behaviour without modification of existing code
- modularity

can be achieved by designing classes such that

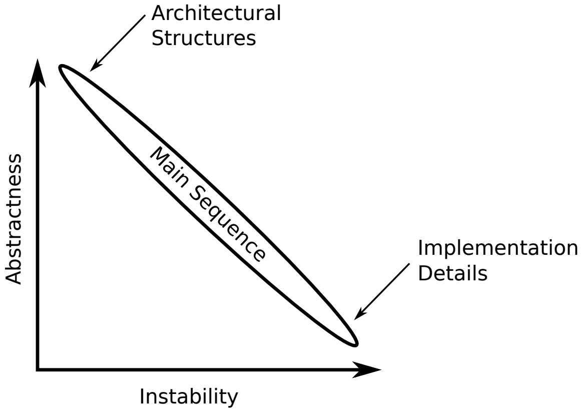
- details depend on generalities
- generalities depend on nothing

$\Rightarrow$ there are (approximately) two types of classes:

1. **abstract base classes**
   - many classes depend on them $\Rightarrow$ very **responsible** (if they change, many other classes need to be changed as well)
   - unlikely to change since they don't depend on anything $\Rightarrow$ very **stable**
2. **concrete classes with implementation details**
   - very few classes depend on them $\Rightarrow$ **irresponsible**
   - likely to change $\Rightarrow$ **instable**

Software package metrics — the main sequence

In an optimal design classes/ categories lie on the "main sequence":



Software package metrics

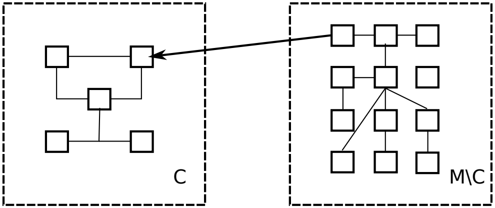
Definition

Let $M = \{a_1, \ldots, a_N\}$ be a set of classes. Let R be the number of class relationships within a subset $O \subset M$. Then the **cohesion** $H(O)$ is defined as (see [Mar95])

$$H(O) := \frac{R+1}{N}$$

Definition

A **category** C is a subset $C \subset M$ with maximum cohesion and minimum relations to $M \backslash C$.



Software package metrics

Definition

Let $C \subset M$ be a category.

Afferent coupling	$Ca(C) :=$ #of classes $a_i \in M \setminus C$ that depend on C
Efferent coupling	$Ce(C) :=$ #of dependencies on classes $a_i \in M \setminus C$
Instability	$I(C) := \dfrac{Ce}{Ce + Ca} \in [0,1]$
Abstractness	$A(C) := \dfrac{\#\text{of abstract classes}}{\#\text{of classes}} \in [0,1]$
Distance from Main Sequence	$D'(C) := \lvert A + I - 1 \rvert \in [0,1]$

Recommendations for metric values in C++

metric	recommended value	reference
SLOC per function	≤ 25	"25 lines 80 columns" rule
NLOCom / SLOC	≈ 1	Arnd Bäcker in Computational Physics
MVG	≤ 10 (simple program)	[McC76]
IFC	?	
WMC	≤ 7	[McC05]
DIT	≤ 3	[CK94]
NOC	≤ 15	[CK94]
CBO	≤ 10	[CK94]
Cohesion $H(C)$	> 0.75	[Mar95]
Distance $D'(C)$	< 0.4	[Mar95]

Program complexity from McCabe's cyclomatic complexity:

$M \leq 10$	simple program
$10 < M \leq 20$	more complex program
$20 < M \leq 40$	very complex program
$M > 40$	untestable program

Literature

S.R. Chidamber and C.F. Kemerer.
A metrics suite for object oriented design.
IEEE Transactions on Software Engineering, 20:6:476–493, 1994.

Robert C. Martin.
Designing Object-Oriented C++ Applications Using The Booch Method.
Prentice-Hall, 1st edition, 1995.

Thomas J. McCabe.
A complexity measure.
In *Proceedings of the 2nd international conference on Software engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

Steve McConnell.
Code Complete.
Microsoft Press, Deutschland, 1st edition, 2005.