

Table of Contents

Average Number of Blank Lines	5
Average Number of Lines of Code	5
Average Number of Lines with Comments	6
Blank Lines of Code	7
Lines of Code	7
Lines with Comments	8
Average Cyclomatic Complexity	9
Average Modified Cyclomatic Complexity	10
Average Strict Cyclomatic Complexity	11
Average Essential Cyclomatic Complexity	12
Average Essential Strict Modified Complexity	12
Average Number of Lines	12
Average Number of Blank Lines	13
Average Number of Lines of Code	14
Average Number of Lines with Comments	15
Blank Lines of Code	16
Lines of Code	17
Lines of Comments	17
Average Cyclomatic Complexity	18
Average Modified Cyclomatic Complexity	19
Average Strict Cyclomatic Complexity	20
Average Essential Cyclomatic Complexity	21
Average Essential Strict Modified Complexity	22
Average Number of Lines	22
Average Number of Blank Lines	23
Average Number of Lines of Code	23
Average Number of Lines with Comments	24
Base Classes	25
Coupling Between Objects	26
Number of Children	27
Classes	27
Class Methods	28

Class Variables.....	28
Executable Unit	29
Number of Files	29
Function	30
Instance Methods	30
Instance Variables	31
Internal Instance Variables	32
Private Instance Variables.....	32
Protected Instance Variables	32
Protected Internal Instance Variables	33
Public Instance Variables	33
Local Methods.....	34
Methods	35
Local Const Methods.....	35
Local Default Visibility Methods	36
Friend Methods.....	36
Local Internal Methods	37
Private Methods.....	37
Protected Methods	38
Local Protected Internal Methods	39
Public Methods	39
Local strict private methods	39
Local strict published methods	40
Modules	40
Program Units	40
Properties.....	40
Auto Implemented Properties	41
Subprograms	41
Inputs	41
Physical Lines	42
Blank Lines of Code	43
Blank html lines.....	45
Blank javascript lines.....	45
Blank php lines	45

Source Lines of Code.....	45
Declarative Lines of Code.....	46
Executable Lines of Code	47
Javascript source code lines.....	48
PHP Source Code Lines.....	48
Lines with Comments.....	49
Blank php lines	50
Source Lines of Code.....	50
Declarative Lines of Code.....	51
Executable Lines of Code	52
Javascript source code lines.....	53
PHP Source Code Lines.....	53
Lines with Comments.....	54
HTML Comment Lines.....	55
Javascript Comment Lines.....	55
PHP Comment Lines.....	55
Inactive Lines.....	55
Preprocessor Lines	56
HTML Lines.....	57
Javascript Lines	57
PHP Lines.....	58
Outputs	58
Coupled Packages	59
Paths.....	59
Paths Log(x).....	59
Semicolons	60
Statements.....	61
Declarative Statements.....	62
Javascript Declarative Statements	64
PHP Declarative Statements	64
Empty Statements.....	64
Javascript Executable Statements.....	65
Cyclomatic Complexity.....	66
Modified Cyclomatic Complexity	67

Strict Cyclomatic Complexity	69
Essential Complexity	70
Essential Strict Modified Complexity	71
Knots	72
Max Cyclomatic Complexity.....	72
Max Modified Cyclomatic Complexity	73
Max Strict Cyclomatic Complexity	74
Max Essential Complexity	74
Max Knots	75
Max Essential Strict Modified Complexity	76
Depth of Inheritance Tree.....	76
Nesting	77
Minimum Knots.....	79
Lack of Cohesion in Methods.....	79
Comment to Code Ratio.....	80
Sum Cyclomatic Complexity.....	81
Sum Modified Cyclomatic Complexity	82
Sum Strict Cyclomatic Complexity	83
Sum Essential Complexity	83
Sum Essential Strict Modified Complexity	84

Average Number of Blank Lines

API Name: AltAvgLineBlank

Description: (Include Inactive) Average number of blank lines for all nested functions or methods, including inactive regions.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union

AltAvgLineBlank				
Formula: SUM(AltCountLineBlank of each function in scope) / # of functions Result (for file): 2 Result (for class): 1				
SayHello::SayHello()	0	SUM=	2 /	2 = 1
SayHello::printHello()	2			
cyclomaticDemo()	2	SUM =	9 /	4 = 2.25
main()	5			

of functions in class
 (Class) AltAvgLineBlank
 (File) AltAvgLineBlank
 # of File Functions (CountDeclFunction)

Average Number of Lines of Code

API Name: AltAvgLineCode

Description: (Include Inactive) Average number of lines containing source code for all nested functions or methods, including inactive regions.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union

AltAvgLineCode				
Formula: SUM(AltCountLineCode of each function in scope) / # of functions				
Result (for file): 15				
Result (for class): 7				
SayHello::SayHello()	1	SUM=	15 /	2 =
SayHello::printHello()	14			
cyclomaticDemo()	27	SUM =	60 /	4 =
main()	18			
<div># of functions in class</div> <div>(Class) AltAvgLineCode</div> <div>(File) AltAvgLineCode</div> <div># of File Functions (CountDeclFunction)</div>				

Average Number of Lines with Comments

API Name: AltAvgLineComment

Description: (Include Inactive) Average number of lines containing comment for all nested functions or methods, including inactive regions.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union

AltAvgLineComment				
Formula: SUM(AltCountLineComment of each function in scope) / # of functions				
Result (for file):1				
Result (for class): 1				
SayHello::SayHello()	0	SUM=	2 /	2 =
SayHello::printHello()	2			
cyclomaticDemo()	0	SUM =	2 /	4 =
main()	0			
<div># of functions in class</div> <div>(Class) AltAvgLineComment</div> <div>(File) AltAvgLineComment</div> <div># of File Functions (CountDeclFunction)</div>				

Blank Lines of Code

API Name: AltCountLineBlank

Description: (Include Inactive) Number of blank lines, including inactive regions.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union

AltCountLineBlank Formula: !(Code Comment Preprocessor) Result (for function printHello()): 2						
	Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){	1	0	0	1	0	0
switch(i){	1	0	0	0	1	0
case 0:	1	0	0	0	1	0
cout << "Hello World" << endl;	1	0	0	0	1	0
case 1:	1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;	1	0	0	0	1	0
default: //A comment here	1	1	0	0	1	0
for(int m=0; m < j; m++);	1	0	0	1	1	0
cout << "hello world" << endl;	1	0	0	0	1	0
}	1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE	0	0	1	0	0	0
cout << "Inactive Line" << endl; // Inactive	1	1	0	0	0	1
#endif	0	0	1	0	0	0
}	0	0	0	0	0	0
}	1	0	0	0	0	0

Lines of Code

API Name: AltCountLineCode

Description: (Include Inactive) Number of lines containing source code, including inactive regions.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union

AltCountLineCode						
Formula: Code Preprocessor						
Result (for function printHello()): 14						
	Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){	1	0	0	1	0	0
switch(i){	1	0	0	0	1	0
case 0:	1	0	0	0	1	0
cout << "Hello World" << endl;	1	0	0	0	1	0
case 1:	1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;	1	0	0	0	1	0
default: //A comment here	1	1	0	0	1	0
for(int m=0; m < j; m++);	1	0	0	1	1	0
cout << "hello world" << endl;	1	0	0	0	1	0
}	1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE	0	0	1	0	0	0
	0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive	1	1	0	0	0	1
#endif	0	0	1	0	0	0
	0	0	0	0	0	0
}	1	0	0	0	0	0

Lines with Comments

API Name: AltCountLineComment

Description: (Include Inactive) Number of lines containing comment, including inactive regions.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union

AltCountLineComment Formula: Comment Result (for function printHello()): 2						
	Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){	1	0	0	1	0	0
switch(i){	1	0	0	0	1	0
case 0:	1	0	0	0	1	0
cout << "Hello World" << endl;	1	0	0	0	1	0
case 1:	1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;	1	0	0	0	1	0
default: //A comment here	1	1	0	0	1	0
for(int m=0; m < j; m++);	1	0	0	1	1	0
cout << "hello world" << endl;	1	0	0	0	1	0
}	1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE	0	0	1	0	0	0
	0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive	1	1	0	0	0	1
#endif	0	0	1	0	0	0
	0	0	0	0	0	0
}	1	0	0	0	0	0

Average Cyclomatic Complexity

API Name: AvgCyclomatic

Description: Average cyclomatic complexity for all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

FORTTRAN: Project, File

Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit

PL/M: Project, File

Python: Project, File, Class

VHDL: Project, File, Architecture

Web: Project, File, PHP Class, PHP Interface

AvgCyclomatic			
Formula: SUM(Cyclomatic of each function in scope) / # of functions Result (for file): 4 Result (for class): 2			
SayHello::SayHello()	1	SUM= 5 / 2 =	2.5
SayHello::printHello()	4		
cyclomaticDemo()	10	SUM = 17 / 4 =	4.25
main()	2		

of functions in class
 (Class) AvgCyclomatic
 (File) AvgCyclomatic
 # of File Functions (CountDeclFunction)

Average Modified Cyclomatic Complexity

API Name: AvgCyclomaticModified

Description: Average modified cyclomatic complexity for all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

FORTTRAN: Project, File

Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit

PL/M: Project, File

Python: Project, File, Class

VHDL: Project, File, Architecture

Web: Project, File, PHP Class, PHP Interface

AvgCyclomaticModified				
Formula: SUM(CyclomaticModified of each function in scope) / # of functions				
Result (forfile):4				
Result (forclass): 2				
	CyclomaticModified			
SayHello::SayHello()	1	SUM=	4 /	2 = 2
SayHello::printHello()	3			
cyclomaticDemo()	8	SUM =	14 /	4 = 3.5
main()	2			

Average Strict Cyclomatic Complexity

API Name: AvgCyclomaticStrict

Description: Average modified cyclomatic complexity for all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit

Python: Project, File, Class

Web: Project, File, PHP Class, PHP Interface

AvgCyclomaticStrict				
Formula: SUM(CyclomaticStrict of each function in scope) / # of functions				
Result (for file): 5				
Result (for class): 2				
SayHello::SayHello()	1	SUM=	5 /	2 = 2.5
SayHello::printHello()	4			
cyclomaticDemo()	12	SUM =	19 /	4 = 4.75
main()	2			
<div># of functions in class</div> <div>(Class) AvgCyclomaticStrict</div> <div>(File) AvgCyclomaticStrict</div> <div># of File Functions (CountDeclFunction)</div>				

Average Essential Cyclomatic Complexity

API Name: AvgEssential

Description: Average Essential complexity for all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

FORTTRAN: Project, File

Java: Project, File, Class, Interface

Jovial: Project, File

Python: Project, File, Class

Web: Project, File, PHP Class, PHP Interface

AvgEssential					
Formula: SUM(Essential of each function in scope) / # of functions					
Result (for file): 1					
Result (for class): 2					
SayHello::SayHello()	1	SUM=	4	/ 2 =	2
SayHello::printHello()	3				
cyclomaticDemo()	1	SUM =	6	/ 4 =	1.5
main()	1				

Essential

of functions in class

(Class) AvgEssential

(File) AvgEssential

of File Functions (CountDeclFunction)

Average Essential Strict Modified Complexity

API Name: AvgEssentialStrictModified

Description: Average Essential complexity for all nested functions or methods.

Metric Type: Complexity

Available For: Ada: Project, File, Package

Average Number of Lines

API Name: AvgLine

Description: Average number of lines for all nested functions or methods.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

AvgLineBlank			
Formula: SUM(CountLineBlank of each function in scope) / # of functions Result (for file): 2 Result (for class): 0			
SayHello::SayHello()	0	SUM= 1 / 2 = 0.5	# of functions in class (Class) AvgLineBlank
SayHello::printHello()	1		
cyclomaticDemo()	2	SUM = 7 / 4 = 1.75	# of File Functions (CountDeclFunction)
main()	4		

Average Number of Lines of Code

API Name: AvgLineCode

Description: Lorenz & Kidd - Average Method Size (AMS)

Average number of lines containing source code for all nested functions or methods.

Metric Type: Count

Available For C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

FORTRAN: Project, File

Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface

PL/M: Project, File

Python: Project, File, Class

Web: Project, File, PHP Class, PHP Interface

AvgLineCode					
Formula: SUM(CountLineCode of each function in scope) / # of functions					
Result (for file): 13					
Result (for class): 6					
SayHello::SayHello()	1	SUM=	12 /	2 =	6
SayHello::printHello()	11				
cyclomaticDemo()	27	SUM =	53 /	4 =	13.3
main()	14				

CountLineCode

of functions in class

(Class) AvgLineCode

(File) AvgLineCode

of File Functions (CountDeclFunction)

Average Number of Lines with Comments

API Name: AvgLineComment

Description Name: Average number of lines containing comment for all nested functions or methods.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

FORTRAN: Project, File

Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface

PL/M: Project, File

Python: Project, File, Class

Web: Project, File, PHP Class, PHP Interface

AvgLineComment					
Formula: SUM(CountLineComment of each function in scope) / # of functions Result (forfile): 0 Result (for class): 0					
SayHello::SayHello()	0	SUM=	1 /	2 =	0.5
SayHello::printHello()	1				
cyclomaticDemo()	0	SUM =	1 /	4 =	0.25
main()	0				

Blank Lines of Code

API Name: AltCountLineBlank

Description: (Include Inactive) Number of blank lines, including in inactive regions.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

AltCountLineBlank						
Formula: !(Code Comment Preprocessor) Result (for function printHello()): 2						
	Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){	1	0	0	1	0	0
switch(i){	1	0	0	0	1	0
case 0:	1	0	0	0	1	0
cout << "Hello World" << endl;	1	0	0	0	1	0
case 1:	1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;	1	0	0	0	1	0
default: //A comment here	1	1	0	0	1	0
for(int m=0; m < j; m++);	1	0	0	1	1	0
cout << "hello world" << endl;	1	0	0	0	1	0
}	1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE	0	0	1	0	0	0
	0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive	1	1	0	0	0	1
#endif	0	0	1	0	0	0
	0	0	0	0	0	0
}	1	0	0	0	0	0

Lines of Code

API Name: AltCountLineCode

Description: Number of lines containing source code, including inactive regions.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

AltCountLineCode Formula: Code Preprocessor Result (for function printHello()): 14						
	Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){	1	0	0	1	0	0
switch(i){	1	0	0	0	1	0
case 0:	1	0	0	0	1	0
cout << "Hello World" << endl;	1	0	0	0	1	0
case 1:	1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;	1	0	0	0	1	0
default: //A comment here	1	1	0	0	1	0
for(int m=0; m < j; m++);	1	0	0	1	1	0
cout << "hello world" << endl;	1	0	0	0	1	0
}	1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE	0	0	1	0	0	0
cout << "Inactive Line" << endl; // Inactive	1	1	0	0	0	1
#endif	0	0	1	0	0	0
}	0	0	0	0	0	0
}	1	0	0	0	0	0

Lines of Comments

API Name: AltCountLineComment

Description: Number of lines containing comments, including comments within inactive regions.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

AltCountLineComment Formula: Comment Result (for function printHello()): 2						
	Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){	1	0	0	1	0	0
switch(i){	1	0	0	0	1	0
case 0:	1	0	0	0	1	0
cout << "Hello World" << endl;	1	0	0	0	1	0
case 1:	1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;	1	0	0	0	1	0
default: //A comment here	1	1	0	0	1	0
for(int m=0; m < j; m++);	1	0	0	1	1	0
cout << "hello world" << endl;	1	0	0	0	1	0
}	1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE	0	0	1	0	0	0
	0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive	1	1	0	0	0	1
#endif	0	0	1	0	0	0
	0	0	0	0	0	0
}	1	0	0	0	0	0

Average Cyclomatic Complexity

API Name: AvgCyclomatic

Description: Average cyclomatic complexity for all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

FORTRAN: Project, File

Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit

PL/M: Project, File

Python: Project, File, Class

VHDL: Project, File, Architecture

Web: Project, File, PHP Class, PHP Interface

AvgCyclomatic					
Formula: SUM(Cyclomatic of each function in scope) / # of functions Result (forfile): 4 Result (for class): 2					
SayHello::SayHello()	1	SUM=	5 /	2 =	2.5
SayHello::printHello()	4				
cyclomaticDemo()	10	SUM =	17 /	4 =	4.25
main()	2				

of functions in class

(Class) AvgCyclomatic

(File) AvgCyclomatic

of File Functions
(CountDeclFunction)

Average Modified Cyclomatic Complexity

API Name: AvgCyclomaticModified

Description: Average modified cyclomatic complexity for all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

FORTRAN: Project, File

Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit

PL/M: Project, File

Python: Project, File, Class

VHDL: Project, File, Architecture

Web: Project, File, PHP Class, PHP Interface

Coupling Between Objects

API Name: CountClassCoupled

Research Name: Chidamber & Kemerer - Coupling Between Objects (CBO)

Description: Number of other classes coupled to. [aka CBO (coupling between object classes)]

The Coupling Between Object Classes (CBO) measure for a class is a count of the number of other classes to which it is coupled. Class A is coupled to class B if class A uses a type, data, or member from class B.

This metric is also referred to as Efferent Coupling (Ce). Any number of couplings to a given class counts as 1 towards the metric total Chidamber & Kemerer suggest that: 1) Excessive coupling between object classes is detrimental to modular design and prevents reuse. 2) Inter-object class couples should be kept to a minimum. 3) The higher the inter-object class coupling, the more rigorous testing needs to be.

Metric Type: Object Oriented

Available For: C/C++: Class, Struct, Union

C#: Class, Struct

Basic: Class, Struct

Java: Class, Interface

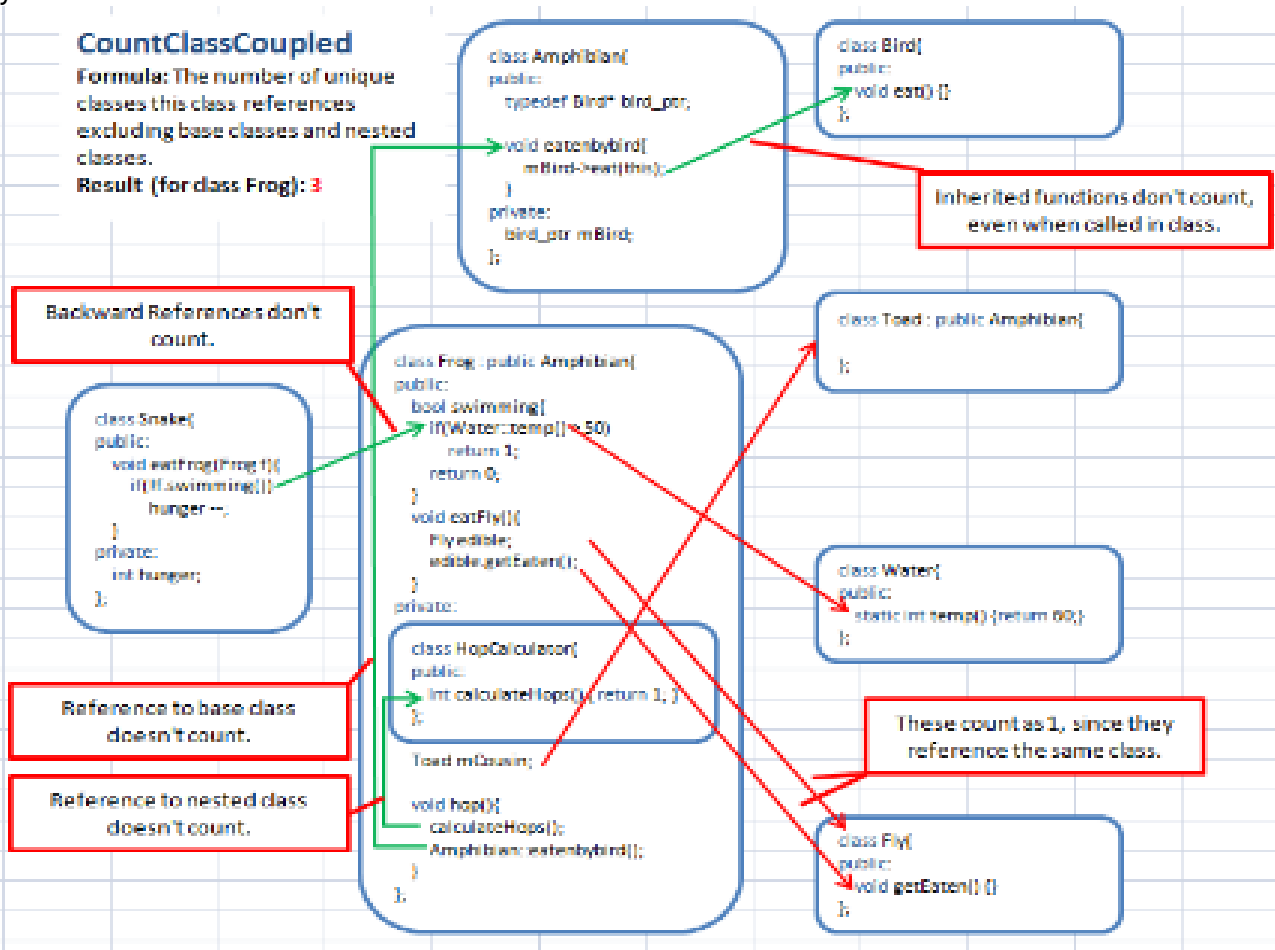
Pascal: Class, Interface

Python: Class

CountClassCoupled

Formula: The number of unique classes this class references excluding base classes and nested classes.

Result (for class Frog): 3



Number of Children

API Name: CountClassDerived

Research Name: Chidamber & Kemerer - Number of Children (NOC)

Description: Number of immediate subclasses. (i.e. the number of classes one level down the inheritance tree from this class).

Metric Type: Object Oriented

Available For: C/C++: Class, Struct, Union

C#: Class, Struct

Basic: Class, Struct

Java: Class, Interface

Pascal: Class, Interface

Python: Class

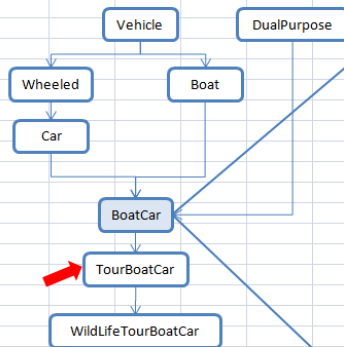
CountClassDerived

Entity Kind:

Reference Kind: "c derive"

Result (for class BoatCar): 1

Referenced Entity	Entity Kind	Reference Kind
metricTestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
mInWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar() : Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers()const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Classes

API Name: CountDeclClass

Number of classes:

Metric Type: Object Oriented

Available For: C/C++: Project, File

C#: Project, File

Basic: Project, File

Java: Project, File

Pascal: Project, File

Python: Project, File

Web: Project, File

Class Methods

API Name: CountDeclClassMethod

Description: Number of class methods.

Metric Type: Object Oriented

Available For: C/C++: Project, Class, Struct, Union

C#: Project, Class, Struct

Basic: Project, Class, Struct

Java: Project, File, Class, Interface

Pascal: Project, Class, Interface

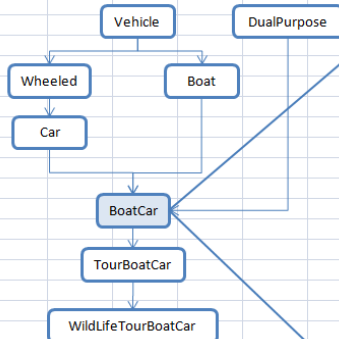
CountDeclClassMethod

Entity Kind: "c memberfunction static"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 2

Referenced Entity	Entity Kind	Reference Kind
metrictestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar(): Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers()const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Class Variables

API Name: CountDeclClassVariable

Research Name: Lorenz & Kidd - Number of Variables (NV)

Description: Number of class variables

Metric Type: Object Oriented

Available For: C/C++: Project, Class, Struct, Union

C#: Project, Class, Struct

Basic: Project, Class, Struct

Java: Project, File, Class, Interface

Pascal: Project, Class, Interface

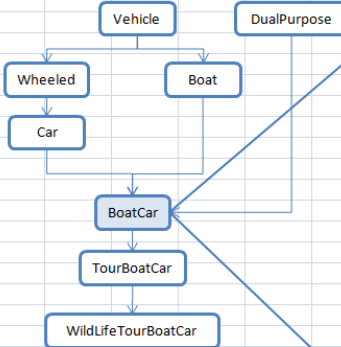
CountDeclClassVariable

Entity Kind: "c member object static"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 1

Referenced Entity	Entity Kind	Reference Kind
metricTestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
mInWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar(): Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers()const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Executable Unit

API Name: CountDeclExecutableUnit**Description:** Number of program units with executable code.**Available For:** C#: Project, File

Ada: Project, File

Basic: Project, File

FORTRAN: Project, File

Java: Project, File

Pascal: Project, File

Python: Project, File

Web: Project, File

Number of Files

API Name: CountDeclFile**Description:** Number of files.**Metric Type:** Count**Available For:** C/C++: Project

C#: Project

Ada: Project

Basic: Project

FORTRAN: Project

Java: Project

Jovial: Project

Pascal: Project

PL/M: Project
Python: Project
VHDL: Project
Web: Project

Function

API Name: CountDeclFunction
Description: Number of functions.
Metric Type: Count
Available For: C/C++: Project, File
C#: Project, File
Java: Project, File
Python: Project, File
Web: Project, File

Instance Methods

API Name: CountDeclInstanceMethod
Research Name: NIM
Description: Number of instance methods - methods defined in a class that are only accessible through an object of that class
Metric Type: Object Oriented
Available For: C/C++: Project, Class, Struct, Union
C#: Project, Class, Struct
Basic: Project, Class, Struct
Java: Project, File, Class, Interface
Pascal: Project, Class, Interface
Python: Project, Class

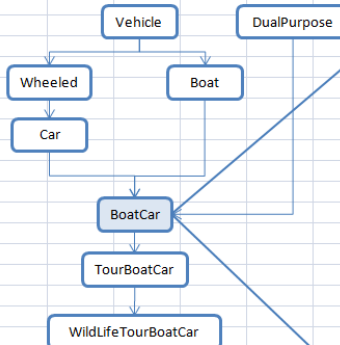
CountDeclInstanceMethod

Entity Kind: "c member function ~static"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 4

Referenced Entity	Entity Kind	Reference Kind
metrictestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar( Car(4), Boat(), minWater(false), mColor("Blue") ) {}
    virtual int passengers() const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Instance Variables

API Name: CountDeclInstanceVariable

Research Name: NIV

Description: Number of instance variables - variables defined in a class that are only accessible through an object of that class

Metric Type: Object Oriented

Available For: C/C++: Project, Class, Struct, Union

C#: Project, Class, Struct

Basic: Project, Class, Struct

Java: Project, File, Class, Interface

Pascal: Project, Class, Interface

Python: Project, Class

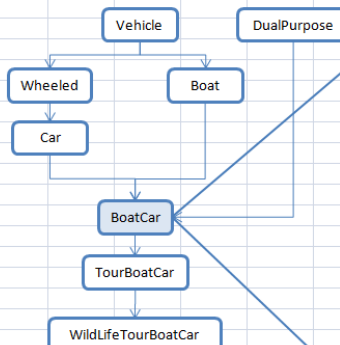
CountDeclInstanceVariablePrivate

Entity Kind: "c private member object ~static"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 1

Referenced Entity	Entity Kind	Reference Kind
metrictestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar( Car(4), Boat(), minWater(false), mColor("Blue") ) {}
    virtual int passengers() const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Internal Instance Variables

API Name: CountDeclInstanceVariableInternal

Description: Number of local protected internal methods.

Metric Type: Count

Available For: C#: Project, Class, Struct

Private Instance Variables

API Name: CountDeclInstanceVariablePrivate

Description: Number of private instance variables.

Metric Type: Object Oriented

Available For: C/C++: Project, Class, Struct, Union

C#: Project, Class, Struct

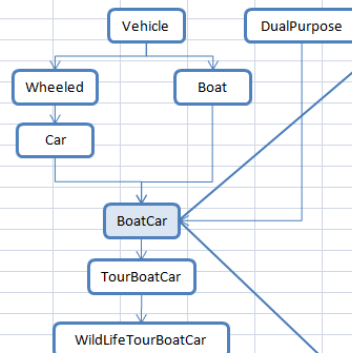
CountDeclInstanceVariablePrivate

Entity Kind: "c private member object ~static"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 1

Referenced Entity	Entity Kind	Reference Kind
metricTestClasses.cpp	C Code File	C DefineIn
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar() : Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers()const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Protected Instance Variables

API Name: CountDeclInstanceVariableProtected

Description: Number of protected instance variables.

Metric Type: Object Oriented

Available For: C/C++: Project, Class, Struct, Union

C#: Project, Class, Struct

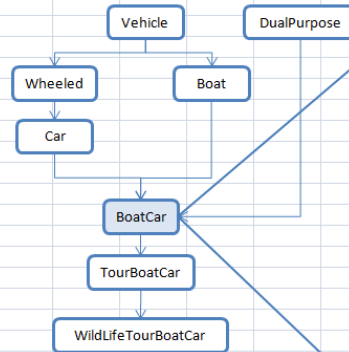
CountDeclInstanceVariableProtected

Entity Kind: "c protected member object ~static"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 1

Referenced Entity	Entity Kind	Reference Kind
metrictestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar() : Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers()const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Protected Internal Instance Variables

API Name: CountDeclInstanceVariableProtectedInternal**Description:** Number of protected internal instance variables.**Metric Type:** Count**Available For:** C#: Project, Class, Struct

Public Instance Variables

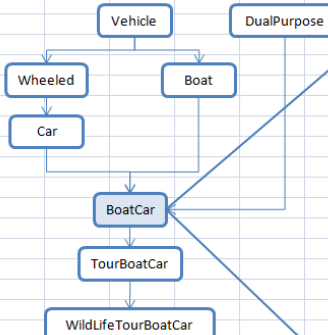
API Name: CountDeclInstanceVariablePublic**Description:** Number of public instance variables.**Metric Type:** Object Oriented**Available For:** C/C++: Project, Class, Struct, Union**C#:** Project, Class, Struct

CountDeclInstanceVariablePublic

Entity Kind: "c public member object ~static"
 Reference Kind: "c declare, c define"

Result (for class BoatCar): 1

Referenced Entity	Entity Kind	Reference Kind
metrictestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar(): Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers()const {return 4;}

    static int numRegistered() {return sRegistered;}
    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Local Methods

API Name: CountDeclMethod

Research Name: Chidamber & Kemerer - Weighted Methods per Class (WMC)

Description Number of local (not inherited) methods.

Metric Type: Object Oriented

Available For: C/C++: Project, Class, Struct, Union

C#: Project, Class, Struct

Basic: Project, Module, Class, Struct

Java: Project, File, Class, Interface

Pascal: Project, Class, Interface

Python: Project, Class

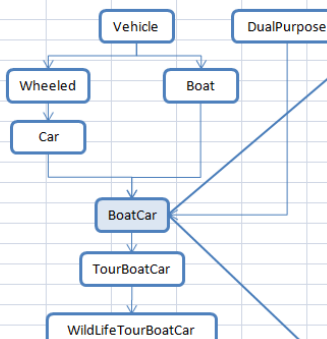
CountDeclMethod

Entity Kind: "c member function ~implicit"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 6

Referenced Entity	Entity Kind	Reference Kind
metrictestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar(): Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers()const {return 4;}

    static int numRegistered() {return sRegistered;}
    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Methods

API Name: CountDeclMethodAll
Research Name: Lorenz & Kidd - Number of Methods (NM)
Description: Number of methods, including inherited ones.
Metric Type: Object Oriented
Available For: C/C++: Project, Class, Struct, Union
C#: Project, Class, Struct
Basic: Project, Class, Struct
Java: Project, Class, Interface
Pascal: Project, Class, Interface
Python: Project, Class

CountDeclMethodAll
Entity Kind: "c member function ~implicit"
Reference Kind: "c declare ~using, c define"
Result (for class BoatCar): **17**

Base Classes	Number of references matching Entity Kind and Reference Kind
Vehicle	1
Wheeled	3
Car	2
Boat	2
Dual Purpose	3
BoatCar	6
	17

```
graph TD
    Vehicle --> Wheeled
    Vehicle --> Boat
    Wheeled --> Car
    Boat --> BoatCar
    Car --> BoatCar
    BoatCar --> TourBoatCar
    BoatCar --> WildLifeTourBoatCar
    DualPurpose --> BoatCar
```

```
class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar() : Car(4), Boat(), mInWater(false), mColor("Blue") {}
    virtual int passengers()const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool mInWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {mInWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
```

Local Const Methods

API Name: CountDeclMethodConst
Description: Number of local const methods.
Metric Type: Object Oriented
Available For: C/C++: Project, Class, Struct, Union

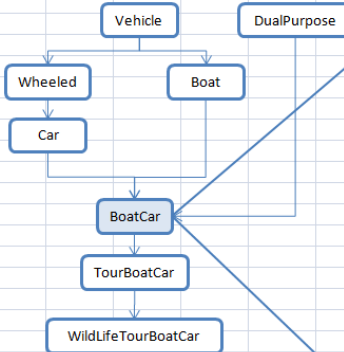
CountDeclMethodConst

Entity Kind: "c member function const ~"implicit"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 1

Referenced Entity	Entity Kind	Reference Kind
metricTestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar(): Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers()const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Local Default Visibility Methods

API Name: CountDeclMethodDefault**Description:** Number of local default visibility methods.**Metric Type:** Object Oriented**Available For:** Java: Project, File, Class, Interface

Friend Methods

API Name: CountDeclMethodFriend**Research Name:** Lorenz & Kidd - Number of Friends (NF), Number of Friend Methods (NFM)**Description:** Number of local (not inherited) friend methods.**Metrics:** Object Oriented**Available For:** C/C++: Project, Class, Struct, Union

CountDeclMethodFriend

Formula: The number of friend functions + the CountDeclMethod of friend classes.

Reference Kind: "cfriend"

Result (for class FriendDemo): 6

Referenced Entity	Entity Kind	Reference Kind
metrictestknots.h	C Code File	C Definein
cohesionClass	C Class Type	C Friend
init	C Unknown Function	C Friend
FriendDemo	C Class Type	C End
FriendDemo	C Class Type	C Endby

```
class FriendDemo{
    friend cohesionClass;

    friend void init();
}
```

CountDeclMethod = 5

Local Internal Methods

API Name: CountDeclMethodInternal

Description: Number of local internal methods.

Metric Type: Count

Available For: C#: Project, Class, Struct

Private Methods

API Name: CountDeclMethodPrivate

Research Name: Number Private Methods (NPRM)

Description: Number of local (not inherited) private methods.

Metric Type: Object Oriented

Available For: C/C++: Project, Class, Struct, Union

C#: Project, Class, Struct

Basic: Project, Module, Class, Struct

Java: Project, File, Class, Interface

Pascal: Project, Class, Interface

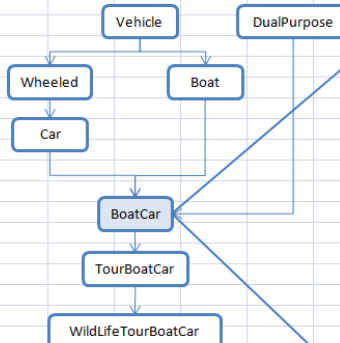
CountDeclMethodPrivate

Entity Kind: "c private member function ~implicit"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 1

Referenced Entity	Entity Kind	Reference Kind
metrictestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar() : Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers() const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Protected Methods

API Name: CountDeclMethodProtected

Description: Number of local protected methods.

Metric Type: Object Oriented

Available For: C/C++: Project, Class, Struct, Union

C#: Project, Class, Struct

Basic: Project, Module, Class, Struct

Java: Project, File, Class, Interface

Pascal: Project, Class, Interface

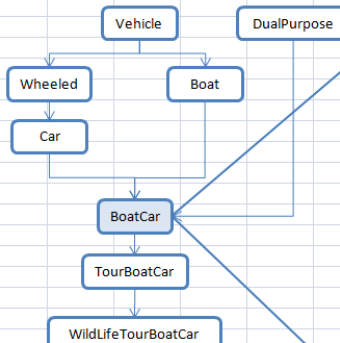
CountDeclMethodProtected

Entity Kind: "c protected member function ~implicit"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 2

Referenced Entity	Entity Kind	Reference Kind
metrictestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar() : Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers() const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Local Protected Internal Methods

API Name: CountDeclMethodProtectedInternal

Description: Number of local protected internal methods.

Metric Type: Count

Available For: C#: Project, Class, Struct

Public Methods

API Name: CountDeclMethodPublic

Research Name: Lorenz & Kidd - Number of Public Methods (PM),NPM

Description: Number of public methods. Only counts local (not inherited) methods.

Metric Type: Object Oriented

Available For: C/C++: Project, Class, Struct, Union

C#: Project, Class, Struct

Basic: Project, Module, Class, Struct

Java: Project, File, Class, Interface

Pascal: Project, Class, Interface

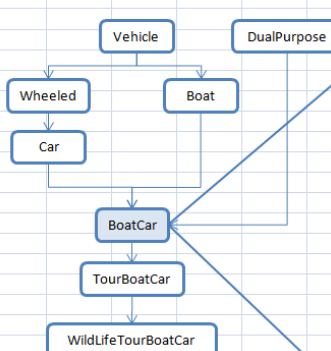
CountDeclMethodPublic

Entity Kind: "c public member function ~implicit"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 3

Referenced Entity	Entity Kind	Reference Kind
metricTestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive



```

class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar() Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers() const {return 4;}
    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
  
```

Local strict private methods

API Name: CountDeclMethodStrictPrivate

Description: Number of local strict private methods.

Metrics Type: Object Oriented Metrics
Available For: Pascal: Project, Class, Interface

Local strict published methods

API Name: CountDeclMethodStrictPublished
Description: Number of local strict published methods.
Metric Type: Object Oriented
Available For: Pascal: Project, Class, Interface

Modules

API Name: CountDeclModule
Description: Number of modules.
Metric Type: Object Oriented
Available For: FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine
Jovial: Project, File
Pascal: Project, File
PL/M: Project

Program Units

API Name: CountDeclProgUnit
Description: Number of program units.
Metric Type: Object Oriented
Available For: FORTRAN: Project, File

Properties

API Name: CountDeclProperty
Description: Number of properties.
Available For: C#: Project, Class, Struct
Pascal: Project, Class, Interface

Auto Implemented Properties

API Name: CountDeclPropertyAuto
Description: Number of auto-implemented properties.
Available For: C#: Project, Class, Struct

CountDeclClassMethod

Entity Kind: "c member function static"

Reference Kind: "c declare, c define"

Result (for class BoatCar): 2

Referenced Entity	Entity Kind	Reference Kind
metrictestClasses.cpp	C Code File	C Definein
Car	C Class Type	C Private Base
Boat	C Class Type	C Public Base
DualPurpose	C Class Type	C Protected Base
BoatCar	C Public Member Function	C Define
passengers	C Public Member Const Function Virtual	C Define
numRegistered	C Public Member Function Static	C Define
minWater	C Public Member Object	C Define
toggleInWater	C Protected Member Function	C Define
mColor	C Protected Member Object	C Define
init	C Function	C Friend
sRegistered	C Unresolved Protected Member Object Static	C Declare
calcSpeed	C Protected Member Function Static	C Define
mMaxPassengers	C Private Member Object	C Define
travel	C Private Member Function	C Define
BoatCar	C Class Type	C End
BoatCar	C Class Type	C Endby
TourBoatCar	C Class Type	C Public Derive

```
graph TD
    Vehicle --> Wheeled
    Vehicle --> Boat
    Wheeled --> Car
    Boat --> BoatCar
    DualPurpose --> BoatCar
    BoatCar --> TourBoatCar
    TourBoatCar --> WildLifeTourBoatCar
```

```
class BoatCar: private Car, public Boat, protected DualPurpose{
public:
    //Public Instance Function
    BoatCar() : Car(4), Boat(), minWater(false), mColor("Blue") {}
    virtual int passengers() const {return 4;}

    static int numRegistered() {return sRegistered;}

    bool minWater; // Public Instance Variable

protected:
    void toggleInWater(bool inWater) {minWater = inWater;}
    char * mColor; // Protected Instance Variable
    friend void init() {}

    static int sRegistered;
    static double calcSpeed(double distance, double time) {
        return distance/time;
    }

private:
    int mMaxPassengers;
    void travel() {}
};
```

Subprograms

API Name: CountDeclSubprogram
Description: Number of subprograms.
Metric Type: Object Oriented
Available For: Ada: Project, File, Package
Basic: Project, File
FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine
Jovial: Project, File, Module, Subroutine
Pascal: Project, File, Compunit, Function, Procedure

Inputs

API Name: CountInput
Research Name: Number of calling subprograms plus global variables read. [aka FANIN]
FANIN The number of inputs a function uses plus the number of unique subprograms calling the function.
Inputs include parameters and global variables that are used in the function, so Functions calledby +

Parameters read + Global Variables read. Of the two general approaches to calculating FANIN (informational versus structural) ours is the informational approach.

Metric Type: Count

Available For: C/C++: Function

C#: Method

FORTRAN: Function, Program, Subroutine

Java: Method

CountInput

Entity Kind: "c global object, c local object, c member object, c parameter, c function"

Reference Kind: "c use ~ptr~inactive, c callby ~inactive"

Notes: recursive function calls and local variables that are not class static variables are not included.

Result (for class function inOutFunc): 6

Referenced Entity	Entity Kind	Reference Kind
metrictestknots.h	C Code File	C Definein
in1	C Parameter	C Define
in2	C Parameter	C Define
inout1	C Parameter	C Define
inout2	C Parameter	C Define
out1	C Parameter	C Define
out2	C Parameter	C Define
in	C Object Global	C Use
in1	C Parameter	C Use
in2	C Parameter	C Use
inout1	C Parameter	C Deref Use
inout2	C Parameter	C Use
out	C Object Global	C Set
in1	C Parameter	C Use
inout1	C Parameter	C Deref Set
in2	C Parameter	C Use
inout2	C Parameter	C Set
in1	C Parameter	C Use
out1	C Parameter	C Deref Set
in2	C Parameter	C Use
out2	C Parameter	C Set
somefunc	C Unknown Function	C Call
in1	C Parameter	C Set
in2	C Parameter	C Set
randomint	C Object Local	C Define
randomint	C Object Local	C Set Init
randomint	C Object Local	C Use
in1	C Parameter	C Set
inOutFunc	C Function	C End
inOutFunc	C Function	C Endby
callingfunc	C Function	C Callby

```

int in = 1;
int out = 1;

int inOutFunc(int in1, int in2, int *inout1, int & inout2, int * out1, int & out2){
    out = in + in1 + in2 + *inout1 + inout2;

    *inout1 = in1;
    inout2 = in2;

    *out1 = in1;
    out2 = in2;

    in1 = somefunc();
    in2 = 2;

    int randomint = 3;
    in1 = randomint;

    return 4;
}

void callingfunc(){
    int a,b,c,d;
    int myval = inOutFunc(1,2,a,b,c,d);
    int temp = out;
    int l = a + b + c + d;
}

```

These are all repeats of entities already counted above.

Local and not class static, so is excluded

Physical Lines

API Name: CountLine

Research Name: NL

Description: Number of physical lines.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class

FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File, Module, Subroutine

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

PL/M: Project, File, Procedure

Python: Project, File, Class, Function

VHDL: Project, File, Procedure, Function, Process, Architecture

Web: Project, File, PHP Class, PHP Interface

	CountLine						
	Formula: $\text{endline} - \text{startline} + 1$						
	Result (for function printHello()): 16						
91		Start Line (Line 92)					Inactive
92	void SayHello::printHello(){	1	0	0	1	0	0
93	switch(i){	1	0	0	0	1	0
94	case 0:	1	0	0	0	1	0
95	cout << "Hello World" << endl;	1	0	0	0	1	0
96	case 1:	1	0	0	0	1	0
97	cout << "HELLO WORLD!" << endl;	1	0	0	0	1	0
98	default: //A comment here	1	1	0	0	1	0
99	for(int m=0; m < j; m++);	1	0	0	1	1	0
100	cout << "hello world" << endl;	1	0	0	0	1	0
101	}	1	0	0	0	0	0
102	#ifdef A_VERY_NICE_VARIABLE	0	0	1	0	0	0
103		0	0	0	0	0	1
104	cout << "Inactive Line" << endl; // Inactive	1	1	0	0	0	1
105	#endif						0
106							0
107	}	1	0	0	0	0	0

Blank Lines of Code

API Name: CountLineBlank

Research Name: BLOC

Description: Number of blank lines.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class

FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File, Module, Subroutine

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

PL/M: Project, File, Procedure

Python: Project, File, Class, Function

VHDL: Project, File, Procedure, Function, Process, Architecture

Web: Project, File, PHP Class, PHP Interface

CountLineBlank							
Formula: !(Code Comment Preprocessor Inactive)							
Result (for function printHello()): 1							
		Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){		1	0	0	1	0	0
switch(i){		1	0	0	0	1	0
case 0:		1	0	0	0	1	0
cout << "Hello World" << endl;		1	0	0	0	1	0
case 1:		1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;		1	0	0	0	1	0
default: //A comment here		1	1	0	0	1	0
for(int m=0; m < j; m++);		1	0	0	1	1	0
cout << "hello world" << endl;		1	0	0	0	1	0
}		1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE	Blank Line, but inactive	0	0	1	0	0	0
		0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive		1	1	0	0	0	1
#endif		0	0	1	0	0	0
		0	0	0	0	0	0
}		1	0	0	0	0	0

Blank html lines

API Name: CountLineBlank_Html

Description: Number of blank html lines.

Metric Type: Count

Available For: Web: Project, File

Blank javascript lines

API Name: CountLineBlank_Javascript

Description: Number of blank javascript lines.

Metric Type: Count

Available For: Web: Project, File

Blank php lines

API Name: CountLineBlank_Php

Description: Number of blank php lines.

Metric Type: Count

Available For: Web: Project, File, PHP Class, PHP Interface

Source Lines of Code

API Name: CountLineCode

Research Name: LOC, SLOC

Description: Number of lines containing source code. [aka LOC] The number of lines that contain source code. Note that a line can contain source and a comment and thus count towards multiple metrics. For Classes this is the sum of the CountLineCode for the member functions of the class.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class

FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File, Module, Subroutine

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

PL/M: Project, File, Procedure

Python: Project, File, Class, Function

VHDL: Project, File, Procedure, Function, Process, Architecture
 Web: Project, File, PHP Class, PHP Interface

CountLineCode							
Formula: Code && ! Inactive							
Result (for function printHello()): 11							
		Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){		1	0	0	1	0	0
switch(i){		1	0	0	0	1	0
case 0:		1	0	0	0	1	0
cout << "Hello World" << endl;		1	0	0	0	1	0
case 1:		1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;		1	0	0	0	1	0
default: //A comment here		1	1	0	0	1	0
for(int m=0; m < j; m++);		1	0	0	1	1	0
cout << "hello world" << endl;		1	0	0	0	1	0
}		1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE		0	0	1	0	0	0
		0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive		1	1	0	0	0	1
#endif		0	0	1	0	0	0
		0	0	0	0	0	0
}		1	0	0	0	0	0

Declarative Lines of Code

API Name: CountLineCodeDecl

Description: Number of lines containing declarative source code. Note that a line can be declarative and executable - int i =0; for instance.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

FORTTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

Python: Project, File, Class, Function

CountLineCodeDecl		Formula: Code && Declarative					
Result (for function printHello()): 2							
		Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){		1	0	0	1	0	0
switch(i){		1	0	0	0	1	0
case 0:		1	0	0	0	1	0
cout << "Hello World" << endl;		1	0	0	0	1	0
case 1:		1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;		1	0	0	0	1	0
default: //A comment here		1	1	0	0	1	0
for(int m=0; m < j; m++);		1	0	0	1	1	0
cout << "hello world" << endl;		1	0	0	0	1	0
}		1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE		0	0	1	0	0	0
		0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive		1	1	0	0	0	1
#endif		0	0	1	0	0	0
		0	0	0	0	0	0
}		1	0	0	0	0	0

Executable Lines of Code

API Name: CountLineCodeExe

Number of lines containing executable source code.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

Python: Project, File, Class, Function

CountLineCodeExe		Formula: Code && Executable Result (for function printHello()): 8					
		Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){		1	0	0	1	0	0
switch(i){		1	0	0	0	1	0
case 0:		1	0	0	0	1	0
cout << "Hello World" << endl;		1	0	0	0	1	0
case 1:		1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;		1	0	0	0	1	0
default: //A comment here		1	1	0	0	1	0
for(int m=0; m < j; m++);		1	0	0	1	1	0
cout << "hello world" << endl;		1	0	0	0	1	0
}		1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE		0	0	1	0	0	0
		0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive		1	1	0	0	0	1
#endif		0	0	1	0	0	0
		0	0	0	0	0	0
}		1	0	0	0	0	0

Javascript source code lines

API Name: CountLineCode_Javascript

Description: Number of javascript lines containing source code.

Metric Type: Count

Available For: Web: Project, File

PHP Source Code Lines

API Name: CountLineCode_Php

Description: Number of php lines containing source code.

Metric Type: Count

Available For: Web: Project, File, PHP Class, PHP Interface

Lines with Comments

API Name: CountLineComment

Research Name: CLOC

Description: Number of lines containing comment. This can overlap with other code counting metrics. For instance, `int j = 1; // comment` has a comment, is a source line, is an executable source line, and a declarative source line.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class

FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File, Module, Subroutine

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

PL/M: Project, File, Procedure

Python: Project, File, Class, Function

VHDL: Project, File, Procedure, Function, Process, Architecture

Web: Project, File, PHP Class, PHP Interface

CountLineComment		Code	Comment	Preprocessor	Declarative	Executable	Inactive
Formula: Comment && ! Inactive							
Result (for function printHello()): 1							
<code>void SayHello::printHello(){</code>		1	0	0	1	0	0
<code>switch(i){</code>		1	0	0	0	1	0
<code>case 0:</code>		1	0	0	0	1	0
<code>cout << "Hello World" << endl;</code>		1	0	0	0	1	0
<code>case 1:</code>		1	0	0	0	1	0
<code>cout << "HELLO WORLD!" << endl;</code>		1	0	0	0	1	0
<code>default: //A comment here</code>		1	1	0	0	1	0
<code>for(int m=0; m < j; m++);</code>		1	0	0	1	1	0
<code>cout << "hello world" << endl;</code>		1	0	0	0	1	0
<code>}</code>		1	0	0	0	0	0
<code>#ifdef A_VERY_NICE_VARIABLE</code>		0	0	1	0	0	0
		0	0	0	0	0	1
<code>cout << "Inactive Line" << endl; // Inactive</code>		1	1	0	0	0	1
<code>#endif</code>		0	0	1	0	0	0
		0	0	0	0	0	0
<code>}</code>		1	0	0	0	0	0

Blank php lines

API Name: CountLineBlank_Php

Description: Number of blank php lines.

Metric Type: Count

Available For: Web: Project, File, PHP Class, PHP Interface

Source Lines of Code

API Name: CountLineCode

Research Name: LOC, SLOC

Description: The number of lines that contain source code. Note that a line can contain source and a comment and thus count towards multiple metrics. For Classes this is the sum of the CountLineCode for the member functions of the class.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class

FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File, Module, Subroutine

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

PL/M: Project, File, Procedure

Python: Project, File, Class, Function

VHDL: Project, File, Procedure, Function, Process, Architecture

Web: Project, File, PHP Class, PHP Interface

CountLineCode							
Formula: Code && ! Inactive							
Result (for function printHello()): 11							
		Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){		1	0	0	1	0	0
switch(i){		1	0	0	0	1	0
case 0:		1	0	0	0	1	0
cout << "Hello World" << endl;		1	0	0	0	1	0
case 1:		1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;		1	0	0	0	1	0
default: //A comment here		1	1	0	0	1	0
for(int m=0; m < j; m++);		1	0	0	1	1	0
cout << "hello world" << endl;		1	0	0	0	1	0
}		1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE		0	0	1	0	0	0
		0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive		1	1	0	0	0	1
#endif		0	0	1	0	0	0
		0	0	0	0	0	0
}		1	0	0	0	0	0

Declarative Lines of Code

API Name: CountLineCodeDecl

Description: Number of lines containing declarative source code. Note that a line can be declarative and executable - int i =0; for instance.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

Python: Project, File, Class, Function

CountLineCodeDecl						
Formula: Code && Declarative						
Result (for function printHello()):2						
	Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){	1	0	0	1	0	0
switch(i){	1	0	0	0	1	0
case 0:	1	0	0	0	1	0
cout << "Hello World" << endl;	1	0	0	0	1	0
case 1:	1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;	1	0	0	0	1	0
default: //A comment here	1	1	0	0	1	0
for(int m=0; m < j; m++);	1	0	0	1	1	0
cout << "hello world" << endl;	1	0	0	0	1	0
}	1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE	0	0	1	0	0	0
	0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive	1	1	0	0	0	1
#endif	0	0	1	0	0	0
	0	0	0	0	0	0
}	1	0	0	0	0	0

Executable Lines of Code

API Name: CountLineCodeExe

Description: Number of lines containing executable source code.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

Python: Project, File, Class, Function

CountLineCodeExe							
Formula: Code && Executable							
Result (for function printHello()): 8							
		Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){		1	0	0	1	0	0
switch(i){		1	0	0	0	1	0
case 0:		1	0	0	0	1	0
cout << "Hello World" << endl;		1	0	0	0	1	0
case 1:		1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;		1	0	0	0	1	0
default: //A comment here		1	1	0	0	1	0
for(int m=0; m < j; m++);		1	0	0	1	1	0
cout << "hello world" << endl;		1	0	0	0	1	0
}		1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE		0	0	1	0	0	0
		0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive		1	1	0	0	0	1
#endif		0	0	1	0	0	0
		0	0	0	0	0	0
}		1	0	0	0	0	0

Javascript source code lines

API Name: CountLineCode_Javascript

Description: Number of javascript lines containing source code.

Metric Type: Count Metric

Available For: Web: Project, File

PHP Source Code Lines

API Name: CountLineCode_Php

Description: Number of php lines containing source code.

Metric Type: Count

Available For: Web: Project, File, PHP Class, PHP Interface

Lines with Comments

API Name: CountLineComment

Research Name: CLOC

Description: Number of lines containing comment. This can overlap with other code counting metrics. For instance `int j = 1; // comment` has a comment, is a source line, is an executable source line, and a declarative source line.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class

FORTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File, Module, Subroutine

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

PL/M: Project, File, Procedure

Python: Project, File, Class, Function

VHDL: Project, File, Procedure, Function, Process, Architecture

Web: Project, File, PHP Class, PHP Interface

CountLineComment Formula: Comment && ! Inactive Result (for function printHello()): 1		Code	Comment	Preprocessor	Declarative	Executable	Inactive
<code>void SayHello::printHello(){</code>		1	0	0	1	0	0
<code> switch(i){</code>		1	0	0	0	1	0
<code> case 0:</code>		1	0	0	0	1	0
<code> cout << "Hello World" << endl;</code>		1	0	0	0	1	0
<code> case 1:</code>		1	0	0	0	1	0
<code> cout << "HELLO WORLD!" << endl;</code>		1	0	0	0	1	0
<code> default: //A comment here</code>		1	1	0	0	1	0
<code> for(int m=0; m < j; m++);</code>		1	0	0	1	1	0
<code> cout << "hello world" << endl;</code>		1	0	0	0	1	0
<code> }</code>		1	0	0	0	0	0
<code>#ifdef A_VERY_NICE_VARIABLE</code>		0	0	1	0	0	0
		0	0	0	0	0	1
<code> cout << "Inactive Line" << endl; // Inactive</code>		1	1	0	0	0	1
<code>#endif</code>		0	0	1	0	0	0
		0	0	0	0	0	0
<code>}</code>		1	0	0	0	0	0

HTML Comment Lines

API Name: CountLineComment_Html

Description: Number of html lines containing comment.

Metric Type: Count

Available For: Web: Project, File

Javascript Comment Lines

API Name: CountLineComment_Javascript

Description: Number of javascript lines containing comment.

Metric Type: Count

Available For: Web:Project, File

PHP Comment Lines

API Name: CountLineComment_Php

Description: Number of php lines containing comment.

Metric Type: Count

Available For: Web: Project, File, PHP Class, PHP Interface

Inactive Lines

API Name: CountLineInactive

Description: This is the number of lines that are inactive from the view of the preprocessor. In other words, they are on the FALSE side of a #if or #ifdef preprocessor directive.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

CountLineInactive Formula: Inactive Result (for function printHello()): 2						
	Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){	1	0	0	1	0	0
switch(i){	1	0	0	0	1	0
case 0:	1	0	0	0	1	0
cout << "Hello World" << endl;	1	0	0	0	1	0
case 1:	1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;	1	0	0	0	1	0
default: //A comment here	1	1	0	0	1	0
for(int m=0; m < j; m++);	1	0	0	1	1	0
cout << "hello world" << endl;	1	0	0	0	1	0
}	1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE	0	0	1	0	0	0
	0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive	1	1	0	0	0	1
#endif	0	0	1	0	0	0
	0	0	0	0	0	0
}	1	0	0	0	0	0

Preprocessor Lines

API Name: CountLinePreprocessor

Description: Number of preprocessor lines.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

CountLinePreprocessor Formula: Preprocessor Result (for function printHello()): 2		Code	Comment	Preprocessor	Declarative	Executable	Inactive
void SayHello::printHello(){		1	0	0	1	0	0
switch(i){		1	0	0	0	1	0
case 0:		1	0	0	0	1	0
cout << "Hello World" << endl;		1	0	0	0	1	0
case 1:		1	0	0	0	1	0
cout << "HELLO WORLD!" << endl;		1	0	0	0	1	0
default: //A comment here		1	1	0	0	1	0
for(int m=0; m < j; m++);		1	0	0	1	1	0
cout << "hello world" << endl;		1	0	0	0	1	0
}		1	0	0	0	0	0
#ifdef A_VERY_NICE_VARIABLE		0	0	1	0	0	0
		0	0	0	0	0	1
cout << "Inactive Line" << endl; // Inactive		1	1	0	0	0	1
#endif		0	0	1	0	0	0
		0	0	0	0	0	0
}		1	0	0	0	0	0

HTML Lines

API Name: CountLine_Html

Description: Number of all html lines.

Metric Type: Count

Available For: Web: Project, File

Javascript Lines

API Name: CountLine_Javascript

Description: Number of all javascript lines.

Metric Type: Count

Available For: Web: Project, File

PHP Lines

API Name: CountLine_Php

Description: Number of all php lines.

Metric Type: Count

Available For: Web: Project, File, PHP Class, PHP Interface

Outputs

API Name: CountOutput

Research Name: FANOUT

Description: Number of called subprograms plus global variables set. [aka FANOUT]. The number of outputs that are SET. This can be parameters or global variables. So Functions calls + Parameters set/modify + Global Variables set/modify. Of the two general approaches to calculating FANOUT (informational versus structural) ours is the informational approach.

Metric Type: Object Oriented

Available For: C/C++: Function

C#: Method

FORTRAN: Function, Program, Subroutine

Java: Method

CountOutput

Entity Kind: "c global object, c local object, c member object, c parameter, c function"

Reference Kind: "c set, c modify, c call ~inactive"

Notes: recursive function calls, local variables that are not class static variables, and parameters that are pass by value are not included. Also counts non-void return type as 1.

Result (for class function inOutFunc): 7

Referenced Entity	Entity Kind	Reference Kind
metrictestknots.h	C Code File	C Definein
in1	C Parameter	C Define
in2	C Parameter	C Define
inout1	C Parameter	C Define
inout2	C Parameter	C Define
out1	C Parameter	C Define
out2	C Parameter	C Define
in	C Object Global	C Use
in1	C Parameter	C Use
in2	C Parameter	C Use
inout1	C Parameter	C Deref Use
inout2	C Parameter	C Use
out	C Object Global	C Set
in1	C Parameter	C Use
inout1	C Parameter	C Deref Set
in2	C Parameter	C Use
inout2	C Parameter	C Set
in1	C Parameter	C Use
out1	C Parameter	C Deref Set
in2	C Parameter	C Use
out2	C Parameter	C Set
somefunc	C Unknown Function	C Call
in1	C Parameter	C Set
in2	C Parameter	C Set
randomint	C Object Local	C Define
randomint	C Object Local	C Set Init
randomint	C Object Local	C Use
in1	C Parameter	C Set
inOutFunc	C Function	C End
inOutFunc	C Function	C Endby
callingfunc	C Function	C Callby

```

int in = 1;
int out = 1;

int inOutFunc(int in1, int in2, int *inout1, int &inout2, int *out1, int &out2) {
    out = in + in1 + in2 + *inout1 + inout2;

    *inout1 = in1;
    inout2 = in2;

    *out1 = in1;
    out2 = in2;

    in1 = somefunc();
    in2 = 2;

    int randomint = 3;
    in1 = randomint;

    return 4;
}

void callingfunc() {
    int a, b, c, d;
    int myval = inOutFunc(1, 2, a, b, c, d);
    int temp = out;
    int l = a + b + c + d;
}

```

Pass By Value, so excluded

Local and not class static, so is excluded

Coupled Packages

API Name: CountPackageCoupled

Description: Number of other packages coupled to.

Metric Type: Object Oriented

Available For: Ada: Project, Package

Paths

API Name: CountPath

Research Name: NPATH

Description: Number of unique paths though a body of code, not counting abnormal exits or gotos.

Metric Type: Complexity

Available For: C/C++: Project, Function

C#: Project, Method

Ada: Project, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, Method

FORTTRAN: Project, Module, Block Data, Function, Program, Subroutine

Java: Project, Method

Jovial: Project, Subroutine

Pascal: Project, Compunit, Function, Procedure

Python: Project, File, Function

Web: Project, File

Paths Log(x)

API Name: CountPathLog

Description: The base 10 logarithm $\text{Log}(x)$ of the number of unique paths though a body of code, not counting abnormal exits or go tos through the code, truncated to an integer value.

Available For: C/C++: Function

C#: Method

Ada: Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Method

FORTTRAN: Module, Block Data, Function, Program, Subroutine

Java: Method

Jovial: Subroutine

Pascal: Compunit, Function, Procedure

Python: File, Function

Web: File

CountPathLog

Base 10 logarithm of the number of unique paths through the code. code with 1 trillion possible paths (10^{13}) would have a CountPathLog metric of 13.

	<code>void pathDemo(){</code>	Paths:
	<code>if(a)</code>	red-purple
	<code>dothis();</code>	red-blue
	<code>if(b)</code>	green-purple
	<code>dothat();</code>	green-blue
	<code>}</code>	

Semicolons

- API Name:** CountSemicolon
- Description:** Number of semicolons.
- Metric Type:** Count
- Available For:** C/C++: Project, File, Function
C#: Project, File, Class, Method
Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task
Java: Project, File, Class, Interface, Method

CountSemicolon

Formula: # of semicolons

Result (for function printHello()): 6

```
void SayHello::printHello(){
    switch(i){
    case 0:
        cout << "Hello World" << end;
    case 1:
        cout << "HELLO WORLD!" << end;
    default: //A comment here
        for(int m=0; m < ; m++;
        cout << "hello world" << end;
    }
#ifdef A_VERY_NICE_VARIABLE
    cout << "Inactive Line" << endl; // Inactive
#endif
}
```

Statements

API Name: CountStmt

Description: Number of declarative plus executable statements.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class

FORTRAN: Project, File, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File, Module, Subroutine

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

PL/M: Project, File, Procedure

Python: Project, File, Class, Function

VHDL: Project, File

Web: Project, File, PHP Class, PHP Interface

	Executable	Declarative (Function)	Empty				
<div>This statement is declarative, but it only counts in the file/class scope.</div> <code>int main(){</code>	0	0	0				
<code> for(int i=0;</code>	0	1	0				
<code> i< 10;</code>	1	0	0				
<code> i++)</code>	1	0	0				
<code> ;</code>	0	0	1				
<code> </code>	0	0	0				
<code> int j = func();</code>	0	1	0				
<code> int k = 0;</code>	0	1	0				
<code> int l = 1;</code>	0	1	0				
<code> </code>	0	0	0				
<code> int m</code>	0	1	0				
<code> = func();</code>	0	0	0				
<code> int n;</code>	0	1	0				
<code> n = 1;</code>	1	0	0				
<code> </code>	0	0	0				
<code>#ifdef A_VERY_NICE_VARIABLE</code>	0	0	0				
<code> </code>	0	0	0				
<code> int j=0;</code>	0	0	0				
<code> cout<<j<<endl;</code>	0	0	0				
<code>#endif</code>	0	0	0				
<code> </code>	0	0	0				
<code> return 0;</code>	1	0	0				
<code>}</code>	0	0	0				
	4	6	1	=	11		
	CountStmtExe				CountStmt		
	CountStmtDecl				CountStmtEmpty		

Declarative Statements

API Name: CountStmtDecl

Description: Number of declarative statements.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class

FORTTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File, Module, Subroutine

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

PL/M: Project, File, Procedure

Python: Project, File, Class, Function

VHDL: Project, File

Web: Project, File, PHP Class, PHP Interface

	Executable	Declarative (Function)	Empty				
<code>int main(){</code>	0	0	0				
<code> for(int i=0;</code>	0	1	0				
<code> i< 10;</code>	1	0	0				
<code> i++)</code>	1	0	0				
<code> ;</code>	0	0	1				
	0	0	0				
<code> int j = func();</code>	0	1	0				
<code> int k = 0;</code>	0	1	0				
<code> int l = 1;</code>	0	1	0				
	0	0	0				
<code> int m</code>	0	1	0				
<code> = func();</code>	0	0	0				
<code> int n;</code>	0	1	0				
<code> n = 1;</code>	1	0	0				
	0	0	0				
<code>#ifdef A_VERY_NICE_VARIABLE</code>	0	0	0				
	0	0	0				
<code> int j=0;</code>	0	0	0				
<code> cout<<j<<endl;</code>	0	0	0				
<code>#endif</code>	0	0	0				
	0	0	0				
<code> return 0;</code>	1	0	0				
<code>}</code>	0	0	0				
	4	6	1	=	11		
	CountStmtExe	CountStmtDecl	CountStmtEmpty		CountStmt		

This statement is declarative, but it only counts in the file/class scope.

CountStmtExe

CountStmtDecl

CountStmt

CountStmtEmpty

Javascript Declarative Statements

API Name: CountStmtDecl_Javascript

Description: Number of javascript declarative statements.

Metric Type: Count

Available For: Web: Project, File

PHP Declarative Statements

API Name: CountStmtDecl_Php

Description: Number of php declarative statements.

Metric Type: Count

Available For: Web: Project, File, PHP Class, PHP Interface

Empty Statements

API Name: CountStmtEmpty

Description: Number of empty statements.

Metric Type: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

	Executable	Declarative (Function)	Empty				
<div>This statement is declarative, but it only counts in the file/class scope.</div> <code>int main(){</code>	0	0	0				
<code> for(int i=0;</code>	0	1	0				
<code> i< 10;</code>	1	0	0				
<code> i++)</code>	1	0	0				
<code> ;</code>	0	0	1				
<code> </code>	0	0	0				
<code> int j = func();</code>	0	1	0				
<code> int k = 0;</code>	0	1	0				
<code> int l = 1;</code>	0	1	0				
<code> </code>	0	0	0				
<code> int m</code>	0	1	0				
<code> = func();</code>	0	0	0				
<code> int n;</code>	0	1	0				
<code> n = 1;</code>	1	0	0				
<code> </code>	0	0	0				
<code>#ifdef A_VERY_NICE_VARIABLE</code>	0	0	0				
<code> </code>	0	0	0				
<code> int j=0;</code>	0	0	0				
<code> cout<<j<<endl;</code>	0	0	0				
<code>#endif</code>	0	0	0				
<code> </code>	0	0	0				
<code> return 0;</code>	1	0	0				
<code>}</code>	0	0	0				
	4	6	1	=	11		
<div>CountStmtExe</div>							
<div>CountStmtDecl</div>							
<div>CountStmtEmpty</div>							
<div>CountStmt</div>							

Javascript Executable Statements

API Name: CountStmtExe_Javascript

Description: Number of javascript executable statements.

Available For: Web: Project, File

PHP Executable Statements

API Name: CountStmtExe_Php

Description: Number of php executable statements.

Metric Type: Count

Available For: Web: Project, File, PHP Class, PHP Interface

Cyclomatic Complexity

API Name: Cyclomatic

Research Name: McCabe Complexity ($v(G)$)

Description: Cyclomatic complexity. McCabe Cyclomatic complexity as per the original NIST paper on the subject. The cyclomatic complexity of any structured program with only one entrance point and one exit point is equal to the number of decision points contained in that program plus one. Understand counts the keywords for decision points (FOR, WHILE, etc) and then adds 1. For a switch statement, each 'case' is counted as 1. For languages with Macros, the expanded Macro text is also included in the calculation.

Metric Type: Complexity

Available For: C/C++: Project, Function

C#: Project, Method

Ada: Project, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, Method

FORTRAN: Project, Module, Block Data, Function, Program, Subroutine

Java: Project, Method

Jovial: Project, Subroutine

Pascal: Project, Compunit, Function, Procedure

PL/M: Project, Procedure

Python: Project, File, Function

VHDL: Project, Procedure, Function, Process

Web: Project, File

Cyclomatic Formula: case,catch,do,for,if,?,while+1 Result (for function cyclomaticDemo()): 10										
	AND	OR	CATCH	DO	FOR	IF	?	WHILE	SWITCH	CASE
void cyclomaticDemo(){	0	0	0	0	0	0	0	0	0	0
bool a=true,b=true,c=true;	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
if(a (b && c)){	1	1	0	0	0	1	0	0	0	0
while(a? b : c){	0	0	0	0	0	0	1	1	0	0
for(int i=0; i < 10; i ++){	0	0	0	0	1	0	0	0	0	0
switch(i){	0	0	0	0	0	0	0	0	1	0
case 1:	0	0	0	0	0	0	0	0	0	1
case 2:	0	0	0	0	0	0	0	0	0	1
cout<<i<<endl;	0	0	0	0	0	0	0	0	0	0
break;	0	0	0	0	0	0	0	0	0	0
case 5:	0	0	0	0	0	0	0	0	0	1
break;	0	0	0	0	0	0	0	0	0	0
default:	0	0	0	0	0	0	0	0	0	0
cout<<i<<endl;	0	0	0	0	0	0	0	0	0	0
}	0	0	0	0	0	0	0	0	0	0
}	0	0	0	0	0	0	0	0	0	0
}	0	0	0	0	0	0	0	0	0	0
} else{	0	0	0	0	0	0	0	0	0	0
try{	0	0	0	0	0	0	0	0	0	0
do{	0	0	0	1	0	0	0	0	0	0
cout<<a<<b<<c<<endl;	0	0	0	0	0	0	0	0	0	0
}while(a);	0	0	0	0	0	0	0	0	0	0
}	0	0	0	0	0	0	0	0	0	0
catch(...){	0	0	1	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
}	0	0	0	0	0	0	0	0	0	0
}	0	0	0	0	0	0	0	0	0	0
}	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1	3 + 1 = 10

Modified Cyclomatic Complexity

API Name: CyclomaticModified

Description: Modified cyclomatic complexity.

Modified McCabe Cyclomatic complexity. The Cyclomatic Complexity except that each decision in a multi-decision structure (switch in C/Java, Case in Ada, computed Goto and arithmetic if in FORTRAN) statement is not counted and instead the entire multi-way decision structure counts as 1

Available For: C/C++: Project, Function

C#: Project, Method

Strict Cyclomatic Complexity

API Name: CyclomaticStrict

Description: Strict cyclomatic complexity. The Cyclomatic Complexity with logical conjunction and logical and in conditional expressions also adding 1 to the complexity for each of their occurrences. i.e. The statement 'if (a && b || c)' would have a cyclomatic of 1 but a strict cyclomatic of 3

Metric Type: Complexity

Available For: C/C++: Project, Function

C#: Project, Method

Ada: Project, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, Method

Java: Project, Method

Jovial: Project, Subroutine

Pascal: Project, Compunit, Function, Procedure

Python: Project, File, Function

VHDL: Project, Procedure, Function, Process

Web: Project, File

Available For: C/C++: Project, Function

C#: Project, Method

Ada: Project, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, Method

FORTRAN: Project, Module, Block Data, Function, Program, Subroutine

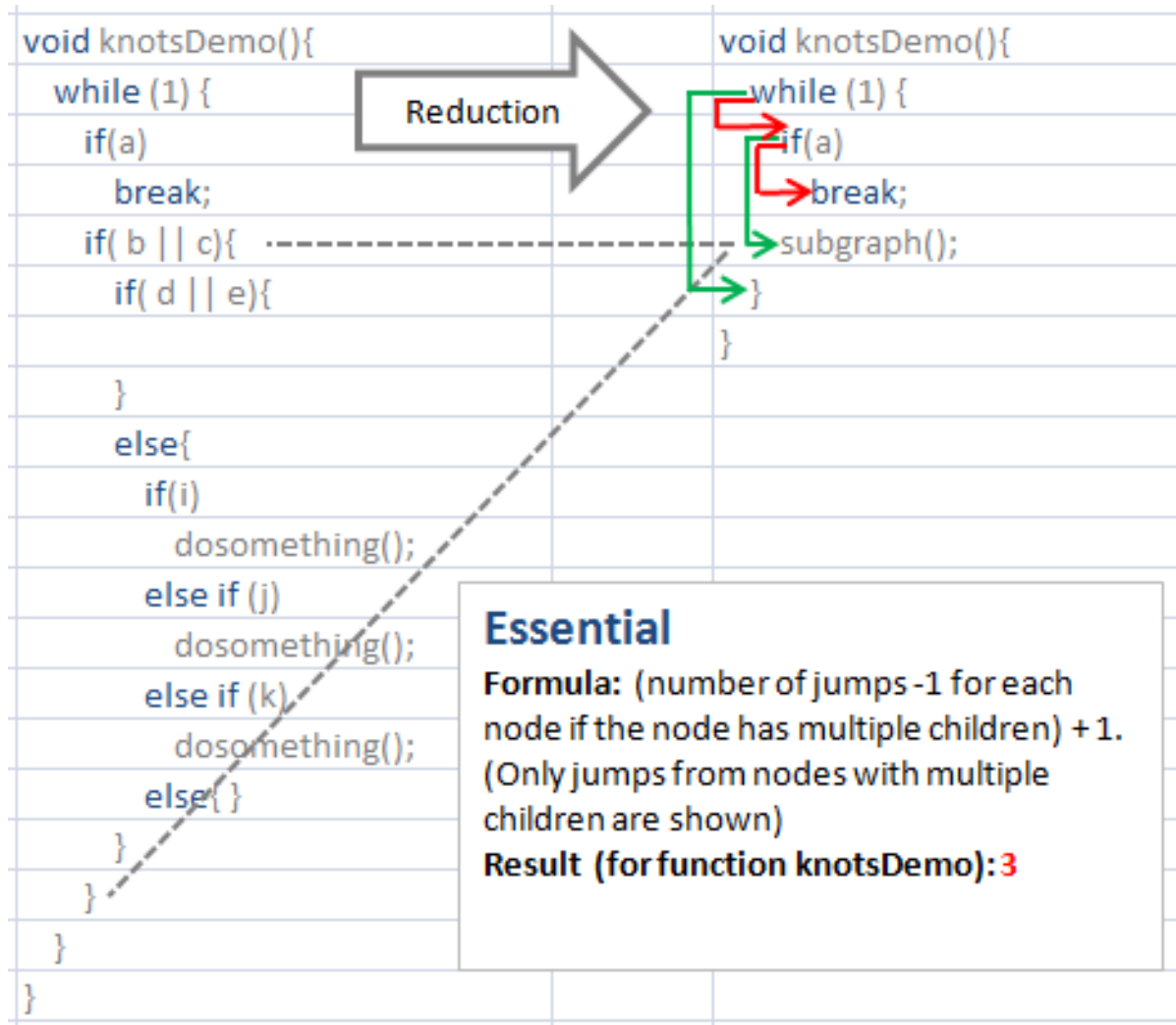
Java: Project, Method

Jovial: Project, Subroutine

Pascal: Project, Compunit, Function, Procedure

Python: Project, File, Function

Web: Project, File



Essential Strict Modified Complexity

API Name: EssentialStrictModified

Description: Strict Modified Essential complexity. The Cyclomatic complexity with short circuit operators (and then/or else) as unstructured but only adds one for all structured paths through case statements after graph reduction.

Metric Type: Complexity

Available For: Ada: Project, Type, Entry, Function, Package, Procedure, Protected, Task

Knots

API Name: Knots

Description: Measure of overlapping jumps. If a piece of code has arrowed lines indicating where every jump in the flow of control occurs, a knot is defined as where two such lines cross each other. The number of knots is proportional to the complexity of the control flow.

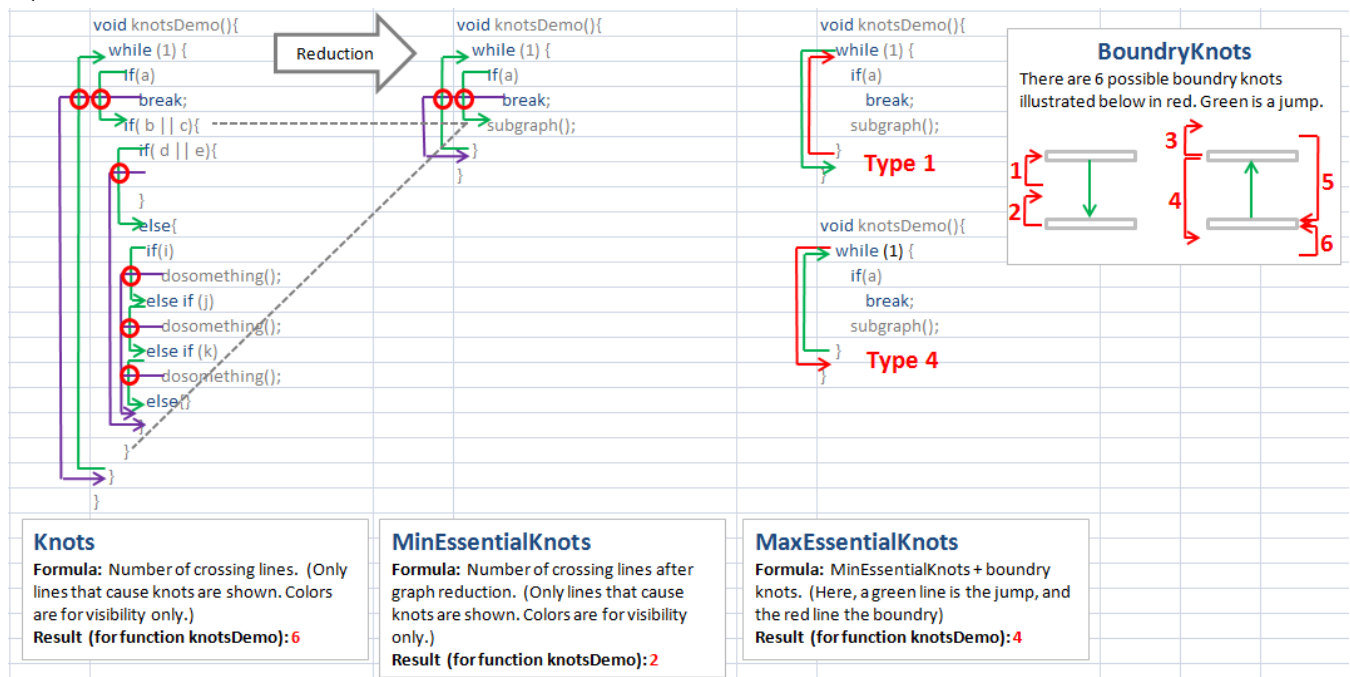
Metric Type: Complexity

Available For: C/C++: Project, Function

C#: Project, Method

Ada: Project, Type, Entry, Function, Package, Procedure, Protected, Task

Java: Project, Method



Max Cyclomatic Complexity

API Name: MaxCyclomatic

Description: Maximum cyclomatic complexity of all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

FORTRAN: Project, File

Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit

Python: Project, File, Class

Web: Project, File, PHP Class, PHP Interface

MaxCyclomatic			
Formula: MAX(Cyclomatic of each function in scope) Result (for file): 10 Result (for class): 4			
SayHello::SayHello()	1	MAX=	4
SayHello::printHello()	4		
cyclomaticDemo()	10	MAX=	10
main()	2		

(Class) SumCyclomatic

(File) SumCyclomatic

Max Modified Cyclomatic Complexity

API Name: MaxCyclomaticModified

Description: Maximum modified cyclomatic complexity of all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

FORTRAN: Project, File

Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit

Python: Project, File, Class

Web: Project, File, PHP Class, PHP Interface

MaxCyclomaticModified Formula: MAX(CyclomaticModified of each function in scope) Result (for file): 8 Result (for class): 3			
	CyclomaticModified		
SayHello::SayHello()	1	MAX=	3
SayHello::printHello()	3		
cyclomaticDemo()	8	MAX=	8
main()	2		

(Class) MaxCyclomaticModified

(File) MaxCyclomaticModified

Max Strict Cyclomatic Complexity

API Name: MaxCyclomaticStrict

Description: Maximum strict cyclomatic complexity of all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit

Python: Project, File, Class

MaxCyclomaticStrict Formula: MAX(CyclomaticStrict of each function in scope) Result (for file): 12 Result (for class): 4			
	CyclomaticStrict		
SayHello::SayHello()	1	MAX=	4
SayHello::printHello()	4		
cyclomaticDemo()	12	MAX=	12
main()	2		

(Class) MaxCyclomaticStrict

(File) MaxCyclomaticStrict

Max Essential Complexity

API Name: MaxEssential

Description: Maximum essential complexity of all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct

Ada: Project, File, Package

Basic: Project, File, Module, Class, Struct

FORTRAN: Project, File

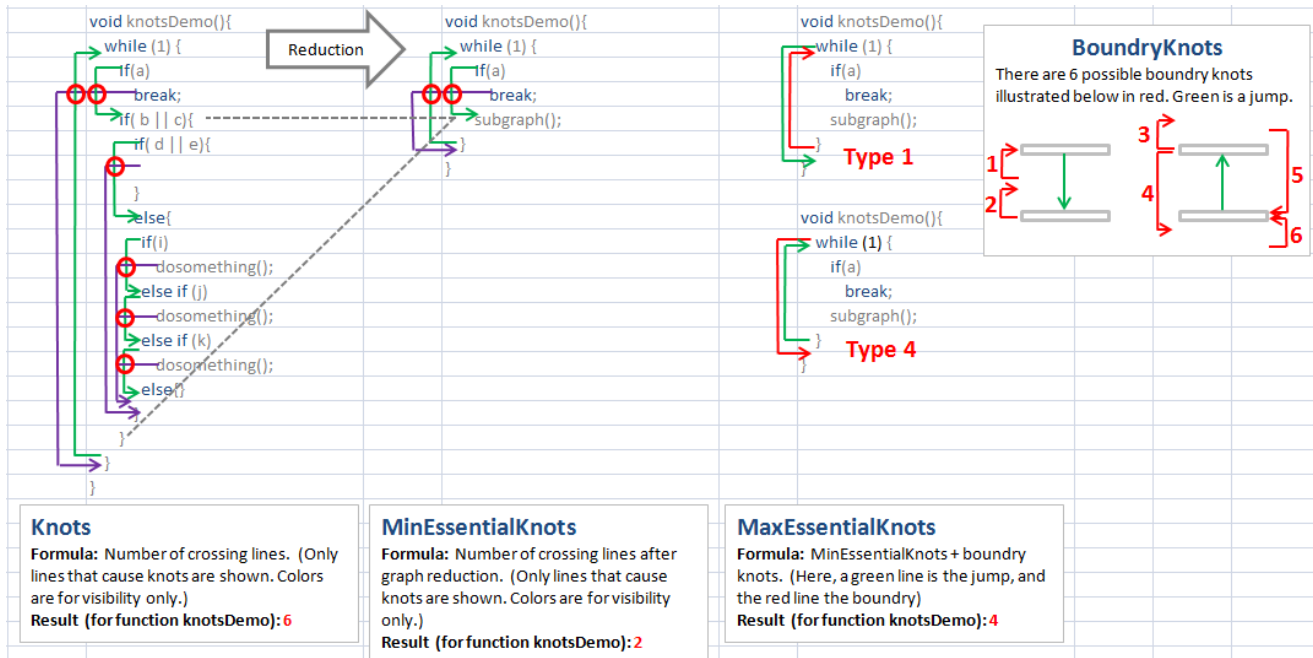
Java: Project, File, Class, Interface

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit

Python: Project, File, Class

Web: Project, File, PHP Class, PHP Interface



Max Knots

API Name: MaxEssentialKnots

Maximum Knots after structured programming constructs have been removed.

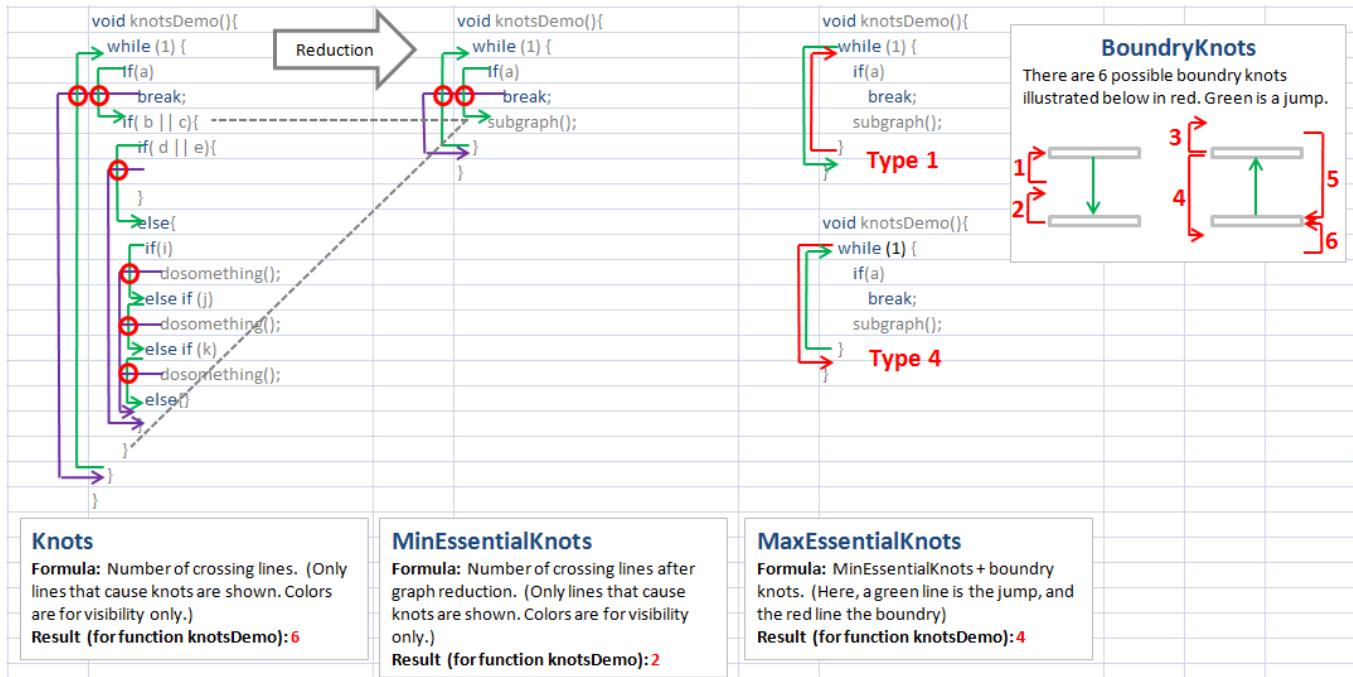
Metric Type: Complexity

Available For: C/C++: Project, Function

C#: Project, Method

Ada: Project, Type, Entry, Function, Package, Procedure, Protected, Task

Java: Project, Method



Max Essential Strict Modified Complexity

API Name: MaxEssentialStrictModified

Description: Maximum strict modified essential complexity of all nested functions or methods.

Available For: Ada: Project, File, Package

Depth of Inheritance Tree

API Name: MaxInheritanceTree

Research Name: Chidamber & Kemerer - Depth of Inheritance Tree (DIT)

Description: Maximum depth of class in inheritance tree. [aka DIT]. The depth of a class within the inheritance hierarchy is the maximum number of nodes from the class node to the root of the inheritance tree. The root node has a DIT of 0. The deeper within the hierarchy, the more methods the class can inherit, increasing its complexity.

Metric Type: Complexity

Available For: C/C++: Project, Class, Struct, Union

C#: Project, Class, Struct

Java: Project, Class, Interface

Pascal: Project, Class, Interface

Python: Project, Class

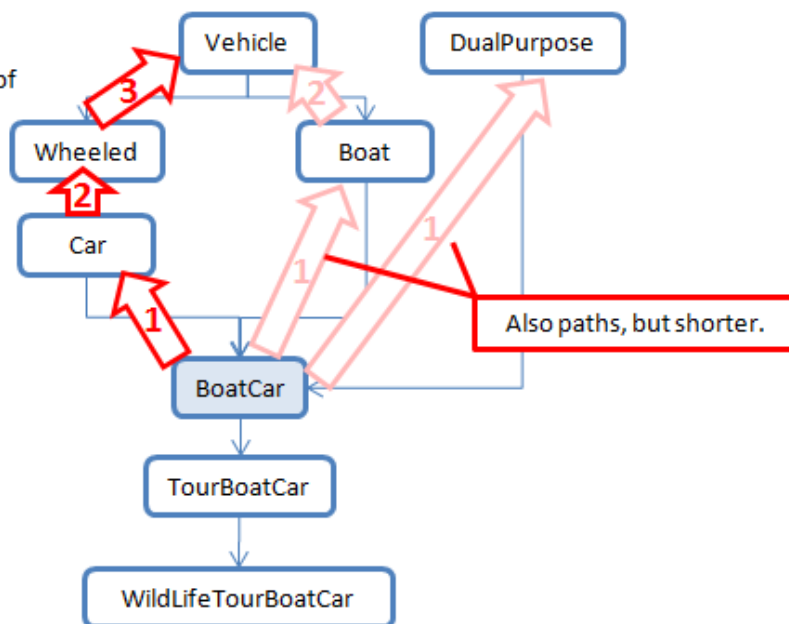
Web: Project, PHP Class, PHP Interface

MaxInheritanceTree

Formula: Longest path to a root of the inheritance tree.

Result (for class Vehicle): 0

Result (for class BoatCar): 3



Nesting

API Name: MaxNesting

Description: Maximum nesting level of control constructs (if, while, for, switch, etc.) in the function.

Metrics: Complexity

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Struct, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class, Struct

FORTTRAN: Project, File, Module, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File, Subroutine

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

Python: Project, File, Class, Function

Web: Project, File, PHP Class, PHP Interface

MaxNesting	
Formula: MAX(nesting)	
Result (for function cyclomaticDemo()): 4	
void cyclomaticDemo(){	0
bool a=true,b=true,c=true;	0
	0
if(a (b && c)){	1
while(a? b : c){	2
for(int i=0; i < 10; i ++){	3
switch(i){	4
case 1:	4
case 2:	4
cout<<i<<endl;	4
break;	4
case 5:	4
break;	4
default:	4
cout<<i<<endl;	4
}	4
}	3
}	2
} else{	1
try{	1
do{	2
cout<<a<<b<<c<<endl;	2
}while(a);	2
}	1
catch(...){	1
	1
}	1
}	1
}	0

Minimum Knots

API Name: MinEssentialKnots

Description: Minimum Knots after structured programming constructs have been removed.

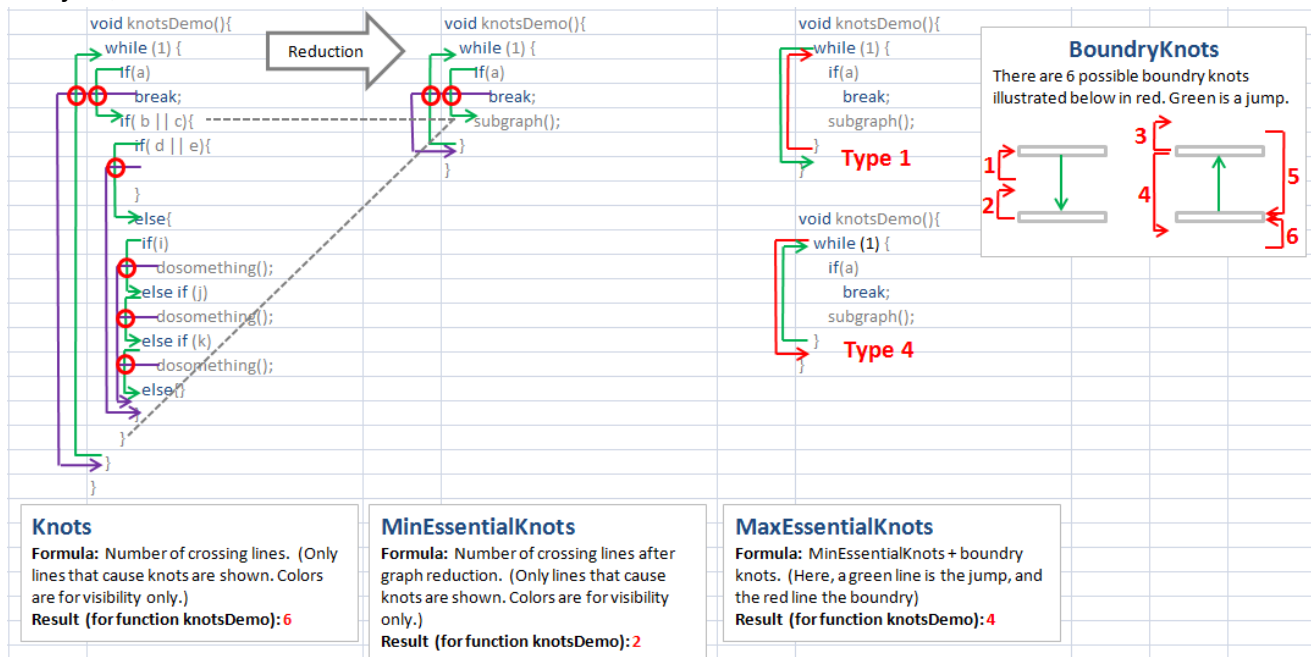
Metrics: Complexity

Available For: C/C++: Project, Function

C#: Project, Method

Ada: Project, Type, Entry, Function, Package, Procedure, Protected, Task

Java: Project, Method



Lack of Cohesion in Methods

API Name: PercentLackOfCohesion

Research Name: Chidamber & Kemerer - Lack of Cohesion in Methods (LCOM/LOCM)

Description: 100% minus average cohesion for class data members. Calculates what percentage of class methods use a given class instance variable. To calculate, average percentages for all of that class's instance variables and subtract from 100%. A lower percentage means higher cohesion between class data and methods.

Metric Type: Object Oriented

Available For: C/C++: Class, Struct, Union

C#: Class, Struct

Ada: Package

Basic: Class, Struct

Java: Class, Interface

Pascal: Class, Interface

PercentLackOfCohesion

Formula: Take each instance variable and divide the number of functions that use it by the total number of functions. Then, average that value for all the instances variables, and subtract from 1. Multiply by 100 to make it a percent.

Result (for class function inOutFunc): 66.67 %

Uses of Variables By Functions

Functions	mVar1	mVar2	sNumObjs
func1	1	1	
func2	1		
func3		1	
func4			
addobj			1
# Funcs That Use Variable	2	2	1
Divided by Total Functions (5)	0.4	0.4	0.2
Averaged Together:	0.3333		
Subtract from 1:	0.6667		
To Percent	66.67%		

```
class cohesionClass{
public:
    void func1(){
        for (int i=0; i<mVar1; ++){
            mVar2 = NULL;
        }
        mVar1 = 3;
    }

    void func2(){
        mVar1 = 4;
    }

    static void addObj(){
        sNumObjs ++
    }

protected:

    void func3(){
        mVar2 = "blue";
    }

private:

    void func4(){

    }

    int mVar1;
    char * mVar2;
    static int sNumObjs = 3;
}
```

Comment to Code Ratio

API Name: RatioCommentToCode

Description: Ratio of number of comment lines to number of code lines. Note that because some lines are both code and comment, this could easily yield percentages higher than 100

Metric: Count

Available For: C/C++: Project, File, Class, Struct, Union, Function

C#: Project, File, Class, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class

FORTRAN: Project, File, Block Data, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File, Module, Subroutine

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

PL/M: Project, File, Procedure

Python: Project, File, Class, Function

VHDL: Project, File, Procedure, Function, Architecture

Web: Project, File

<div> RatioCommentToCode Formula: $\text{CountLineComment} / \text{CountLineCode}$ </div>			
	CountLineComment	CountLineCode	RatioCommentToCode
SayHello::SayHello()	0 /	1 =	0
SayHello::printHello()	1 /	11 =	0.09
cyclomaticDemo()	0 /	27 =	0
main()	0 /	14 =	0

Sum Cyclomatic Complexity

API Name: SumCyclomatic

Research Name: Chidamber & Kemerer - Weighted Methods per Class (WCM)

Description: Sum of cyclomatic complexity of all nested functions or methods. [aka WMC]

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class, Struct

FORTRAN: Project, File, Module, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File

Pascal

Project, File, Class, Interface, Compunit, Function, Procedure

Python

Project, File, Class

Web

Project, File, PHP Class, PHP Interface

Code	Cyclomatic	Sum	Scope
SayHello::SayHello()	1	SUM = 5	(Class) SumCyclomatic
SayHello::printHello()	4		
cyclomaticDemo()	10	SUM = 17	(File) SumCyclomatic
main()	2		

Sum Modified Cyclomatic Complexity

API Name: SumCyclomaticModified

Description: Sum of modified cyclomatic complexity of all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class, Struct

FORTTRAN: Project, File, Module, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File

Pascal: Project, File, Class, Interface, Component, Function, Procedure

Python: Project, File, Class

Web: Project, File, PHP Class, PHP Interface

SumCyclomaticModified	
Formula: SUM(CyclomaticModified of each function in scope)	
Result (for file): 14	
Result (for class): 4	
SayHello::SayHello()	1
SayHello::printHello()	3
cyclomaticDemo()	8
main()	2
SUM = 4	
SUM = 14	

Sum Strict Cyclomatic Complexity

API Name: SumCyclomaticStrict

Description: Sum of strict cyclomatic complexity of all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class, Struct

Java: Project, File, Class, Interface, Method

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

Python: Project, File, Class

Web: Project, File, PHP Class, PHP Interface

SumCyclomaticStrict			
Formula: SUM(CyclomaticStrict of each function in scope) Result (forfile): 19 Result (for class): 5			
SayHello::SayHello()	1	SUM=	5
SayHello::printHello()	4		
cyclomaticDemo()	12	SUM =	19
main()	2		

(Class) AvgCyclomaticStrict

(File) AvgCyclomaticStrict

Sum Essential Complexity

API Name: SumEssential

Description: Sum of essential complexity of all nested functions or methods.

Metric Type: Complexity

Available For: C/C++: Project, File, Class, Struct, Union

C#: Project, File, Class, Struct, Method

Ada: Project, File, Type, Entry, Function, Package, Procedure, Protected, Task

Basic: Project, File, Method, Module, Class, Struct

FORTRAN: Project, File, Module, Function, Program, Subroutine

Java: Project, File, Class, Interface, Method

Jovial: Project, File

Pascal: Project, File, Class, Interface, Compunit, Function, Procedure

Python: Project, File, Class

