
# Essential Metrics Project Manager (EPM)

## Metrics Definitions

*1.1.0.0*

# Contents

# Document

## Purpose

This document describes the Metrics Codes used for the Essential Metrics PM command-line source code metrics tool.

## Readership

This User Guide is intended for end-users, system administrators. Management and non-technical users should refer to our website – http://www.powersoftware.com/epm/

## Version History

| | | |
|---|---|---|
| 1.00.000 | 13-Dec-2007 | Document created. |
| 1.1.0.0 | 08-Sep-2009 | Various metrics added, including PLOC and CRN* metrics. |

# Summary

We use a unique numeric code for each metric, as well as an alpha code which is more descriptive. Here is a list of the IDs, Codes and a brief description of the metric:

| ID | Code | Description | Project | File |
|---|---|---|---|---|
| 100 | LOC | Lines of Code | ✓ | ✓ |
| 101 | SLOC | Source Lines of Code | ✓ | ✓ |
| 102 | SLOC_NAT | Source Native Lines of Code | ✓ | ✓ |
| 103 | SLOC_TAG | Source Tag Lines of Code | ✓ | ✓ |
| 104 | SLOC_HTM | Source HTML Lines of Code | ✓ | ✓ |
| 105 | SLOC_SCR | Source Script Lines of Code | ✓ | ✓ |
| 106 | PLOC | Preprocessor Directive Lines of Code | ✓ | ✓ |
| 107 | LLOC | Logical Lines of Code (semi-colon count – formerly NSC) | ✓ | ✓ |
| 108 | N1 | Total No. of Operators | | ✓ |
| 109 | N2 | Total No. of Operands | | ✓ |
| 110 | n1 | No. of unique or distinct Operators | | ✓ |
| 111 | n2 | No. of unique or distinct Operands | | ✓ |
| 112 | N | Halstead program Length (calculated as N1 + N2) | | ✓ |
| 113 | n | Halstead program Vocabulary (calculated as n1 + n2) | | ✓ |
| 114 | V | Halstead Volume (calculated as V = Nlog2n) | | ✓ |
| 115 | D | Halstead program Difficulty | | ✓ |
| 116 | E | Halstead program Effort (calculated as D * V) | | ✓ |
| 117 | B | Halstead Bug Prediction | | ✓ |
| 118 | J_COM | Java-style Comment Lines | ✓ | ✓ |
| 119 | C_COM | C-style Comment Lines | ✓ | ✓ |
| 120 | EOL_COM | To End of Line Comment Lines | ✓ | ✓ |
| 121 | COM_LOC | Total Comment Lines | ✓ | ✓ |
| 122 | BYTES | File size in bytes | ✓ | ✓ |
| 123 | NFILE | Number of Files | ✓ | |
| 124 | CHG_SLOC | Changed Source Lines of Code | ✓ | ✓ |
| 125 | DEL_SLOC | Deleted Source Lines of Code | ✓ | ✓ |
| 126 | ADD_SLOC | Added Source Lines of Code | ✓ | ✓ |
| 127 | CRN_SLOC | Churn Source Lines of Code | ✓ | ✓ |
| 128 | CHG_LLOC | Changed Logical Lines of Code | ✓ | ✓ |
| 129 | DEL_LLOC | Deleted Logical Lines of Code | ✓ | ✓ |
| 130 | ADD_LLOC | Added Logical Lines of Code | ✓ | ✓ |
| 131 | CRN_LLOC | **Churn** Logical Lines of Code | ✓ | ✓ |
| 132 | CHG_FILE | Changed Files | ✓ | |
| 133 | DEL_FILE | Deleted Files | ✓ | |
| 134 | ADD_FILE | Added Files | ✓ | |
| 135 | CRN_FILE | **Churn Files** | ✓ | |

# Metrics Definitions

## LOC — Lines of code

**Number of Lines in this file, including source, whitespace and comments.**

**Effectively, a count of the number of newlines in the file.**

## SLOC — Source lines of code

**Number of Source lines in this file, excluding whitespace and comments.**

**This is not a count of semicolons or distinct statements, but a count of physical lines that contain source code.**

**If you are interested in "logical" lines, see NSC.**

```
int i=0; float j=0; // This is 1 SLOC

cout << "Testing"
<< " Hello"
<< " there." ; // This is 3 SLOCs
```

## SLOC_NAT — Source Native Lines of Code

**The number of lines of code in this file containing Native Source.**

**Native Source is code that is native to the main language, used when analysing HTML-generating software.**

## SLOC_TAG — Source Tag Lines of Code

**The number of lines of code in this file containing Tags.**

**Tags are used in conjuction with HTML-generating software, such as ASP or JSP.**

## SLOC_HTM — Source HTML Lines of Code

**The number of lines of code in this file containing HTML.**

## SLOC_SCR — Source Script Lines of Code

**The number of lines of code in this file containing Script.**

**Script in this context means Javascript or VBScript used in HTML source, either a HTML file or HTML-generating source.**

## PLOC — Preprocessor Directive Lines of Code

**The number of preprocessor directive lines in the file, typically those that begin with a #. Preprocessor Directive LOC is included as part of SLOC and LOC, but identified separately at the filelevel for clarity.**

## LLOC — Logical Lines of Code

**A count of the number of semicolons in this file excluding those within comments and**

**string literals. This is useful for approximating "logical lines of code". E.g.:**

```
cout << "Hello" << endl ; /* output ; return */
cout << "Hello ; World" << endl ;
```

**LLOC = 2.**

## N1 — Halstead total number of operators

**The total number of operators in the file.**

**N1 is part of the Halstead set, and of little interest on its own. When combined with the other Halstead fundamental metrics, it can be used to determine Volume, Difficult, Effort and Cyclomatic complexity.**

## N2 — Halstead total number of operands

**The total number of operands in the file.**

**N2 is part of the Halstead set, and of little interest on its own. When combined with the other Halstead fundamental metrics, it can be used to determine Volume, Difficult, Effort and Cyclomatic complexity.**

## n1 — Halstead number of unique operators

**The number of unique or distinct operators in the file.**

**n1 is part of the Halstead set, and of little interest on its own. When combined with the other Halstead fundamental metrics, it can be used to determine Volume, Difficult, Effort and Cyclomatic complexity.**

## n2 — Halstead number of unique operands

**The number of unique or distinct operands in the file.**

**n2 is part of the Halstead set, and of little interest on its own. When combined with the other Halstead fundamental metrics, it can be used to determine Volume, Difficult, Effort and Cyclomatic complexity.**

## N — Halstead program Length

**Halstead program Length (N) for this file is calculated as N1 + N2.**

**N1 = Total number of Operators**
**N2 = Total number of Operands**

**Formula N = N1 + N2**

## n — Halstead program Vocabulary

**Halstead program Vocabulary (n) for this file is calculated as n1 + n2.**

**n1 = Number unique or distinct Operators**
**n2 = Number unique or distinct Operands**

**Formula n = n1 + n2**

## V — Halstead program Volume

**Halstead Volume for the file. Calculated as:**

**Formula V = $N\log_2 n$**

## D — Halstead program Difficulty

**A measure of how difficult this file's code is to understand.**

**Formula D = (n1/2) * (N2/n2) Where: n1 - the number of distinct operators n2 - the number of distinct operands N1 - the total number of operators N2 - the total number of operands**

## E — Halstead program Effort

**Effort = Difficulty * Volume for this file.**

**Formula E = D * V**

## B — Halstead Bug Prediction

**This variant of Halstead Effort is a measure of the likelihood of bugs within the file.**

**Formula (N *log*n) / 3000**

## J_COM — Java style comments

**The number of Java style comments in the file.**

**A Java style comment begins `/**` and ends `*/`. E.g.:**

```
/**
 * An interface for handling Application level operations.
 *
 */
public interface ApplicationHandler {
```

**J_COM = 1.**

## C_COM — C style comments

**The number of C style comments in the file.**

**A C style comment begins `/*` and ends `*/`.**

**In the following example C_COM = 1.**

```
int i = 0 ; /* This is a useful
variable here for
many purposes */
```

| EOL_COM | To End-of-Line comments |
|---|---|

**The number of to end-of-line comment lines in this file.**

**This type of comment begins // and finishes at the end of the line.**

**In the following example EOL_COM=2.**

```
int i = 0 ; // Index variable
float pi = 3.14 ; // Handy pi variable
char star = '*' ;
```

| COM_LOC | Comment lines of code |
|---|---|

**The total number of lines of comment in the file.**

| BYTES | Number of bytes |
|---|---|

**The total number of bytes in the file.**

| NFILE | Number of Files |
|---|---|

**The total number of files in the project.**

## CHG_SLOC — Source Lines of Code (Changed)

**Number of source lines that have been changed between the new file and the old file.**

**For example the following would have CHG_SLOC=1.**

**Old File**

```
void functionA(int k) {
	cout << "Parameter: " << k << endl ;
}

void functionB(j) {
	//Unused function
}

int main(int argc, char **argv) {

	int i ;
	int j=100 ;

	for (i=0 ; i<10 ; i++) {

		// Calling functionA()
		functionA(i) ;

		while (j>0) {
			j-= 5 ;
		}

	}
}
```

**New File**

```
void functionA(int k) {
	cout << "Parameter: " << k << endl ;
}

int main(int argc, char **argv) {

	int i ;
	int j=100 ;

	int count=0 ;

	for (i=0 ; i<10 ; i++) {

		// Calling functionA()
		functionA(i) ;

		while (j>0) {
			j-= 10 ;
		}
		while(count<10) {
			count++ ;
		}

	}
}
```

**Color Key**
**Added**
**Changed**
**Deleted**

## DEL_SLOC — Source Lines of Code (Deleted)

**Number of source lines that have been deleted between the new file and the old file.**

**For example the following would have DEL_SLOC=2. (The comment is not a SLOC)**

**Old File**

```
void functionA(int k) {
	cout << "Parameter: " << k << endl ;
}

void functionB(j) {
	//Unused function
}

int main(int argc, char **argv) {

	int i ;
	int j=100 ;

	for (i=0 ; i<10 ; i++) {

		// Calling functionA()
		functionA(i) ;

		while (j>0) {
			j-= 5 ;
		}

	}
}
```

**New File**

```
void functionA(int k) {
	cout << "Parameter: " << k << endl ;
}

int main(int argc, char **argv) {

	int i ;
	int j=100 ;

	int count=0 ;

	for (i=0 ; i<10 ; i++) {

		// Calling functionA()
		functionA(i) ;

		while (j>0) {
			j-= 10 ;
		}
		while(count<10) {
			count++ ;
		}

	}
}
```

**Color Key**
**Added**
**Changed**
**Deleted**

## ADD_SLOC — Source Lines of Code (Added)

**Number of source lines that have been added between the new file and the old file.**

**For example the following would have ADD_SLOC=4.**

**Old File**

```
void functionA(int k) {
	cout << "Parameter: " << k << endl ;
}

void functionB(j) {
	//Unused function
}

int main(int argc, char **argv) {

	int i ;
	int j=100 ;

	for (i=0 ; i<10 ; i++) {

		// Calling functionA()
		functionA(i) ;

		while (j>0) {
			j-= 5 ;
		}

	}
}
```

**New File**

```
void functionA(int k) {
	cout << "Parameter: " << k << endl ;
}

int main(int argc, char **argv) {

	int i ;
	int j=100 ;

	int count=0 ;

	for (i=0 ; i<10 ; i++) {

		// Calling functionA()
		functionA(i) ;

		while (j>0) {
			j-= 10 ;
		}
		while(count<10) {
			count++ ;
		}

	}
}
```

**Color Key**
**Added**
**Changed**
**Deleted**

| CRN_SLOC | Churn Source Lines of Code |
|---|---|

**Churn metrics give an overall indicator of the total "change" in source lines between the new file and the old file.**

**Formula CRN_SLOC = CHG_SLOC + DEL_SLOC + ADD_SLOC**

| CHG_LLOC | Changed Logical Lines of Code |
|---|---|

**The number of changed Logical Lines of Code in this file.**

**Changes are measured using Power Software's implementation of the GNU diff but instead of comparing lines terminated by newlines, comparison is made between "lines" terminated by semi-colons.**

**CHG_LLOC gives you a potentially more accurate estimatation of the extent of changed source code, especially where a single "logical line" spans multiple lines in this file.**

| DEL_LLOC | Deleted Logical Lines of Code |
|---|---|

**The number of deleted Logical Lines of Code in this file.**

**Deleted logical lines are counted using Power Software's implementation of the GNU diff but instead of comparing lines terminated by newlines, comparison is made between "lines" terminated by semi-colons.**

**DEL_LLOC gives you a potentially more accurate estimatation of the extent of deleted lines in your source code, especially where a single "logical line" spans multiple lines in this file.**

| ADD_LLOC | Added Logical Lines of Code |
|---|---|

**The number of deleted Logical Lines of Code in this file.**

**Added logical lines are counted using Power Software's implementation of the GNU diff but instead of comparing lines terminated by newlines, comparison is made between "lines" terminated by semi-colons.**

**ADD_LLOC gives you a potentially more accurate estimatation of the extent of added lines in your source code, especially where a single "logical line" spans multiple lines in this file.**

| CRN_LLOC | Churn Logical Lines of Code |
|---|---|

**Churn metrics give an overall indicator of the total "change" in logical lines between the new file and the old file.**

**Formula CRN_LLOC = CHG_LLOC + DEL_LLOC + ADD_LLOC**

## CHG_FILE — Number of Files (Changed)

**Number of Files that have been changed between the old project and the new project.**

**For example the following would have CHG_FILE=1.**

| Old Project | New Project |
|---|---|
| sourcefile1 | sourcefile1 |
| sourcefile2 | sourcefile2 |
| sourcefile3 | sourcefile4 |

**Color Key**
**Added**
**Changed**
**Deleted**

## DEL_FILE — Number of Files (Deleted)

**Number of Files that have been deleted between the old project and the new project.**

**For example the following would have DEL_FILE=1.**

| Old Project | New Project |
|---|---|
| sourcefile1 | sourcefile1 |
| sourcefile2 | sourcefile2 |
| sourcefile3 | sourcefile4 |

**Color Key**
**Added**
**Changed**
**Deleted**

## ADD_FILE — Number of Files (Added)

**Number of Files that have been added between the old project and the new project.**

**For example the following would have ADD_FILE=1.**

| Old Project | New Project |
|---|---|
| sourcefile1 | sourcefile1 |
| sourcefile2 | sourcefile2 |
| sourcefile3 | sourcefile4 |

**Color Key**
**Added**
**Changed**
**Deleted**

## CRN_FILE — Churn Files

**Churn metrics give an overall indicator of the total "change" between the new project and the old project.**

**Formula CRN_FILE = CHG_FILE + DEL_FILE + ADD_FILE**