

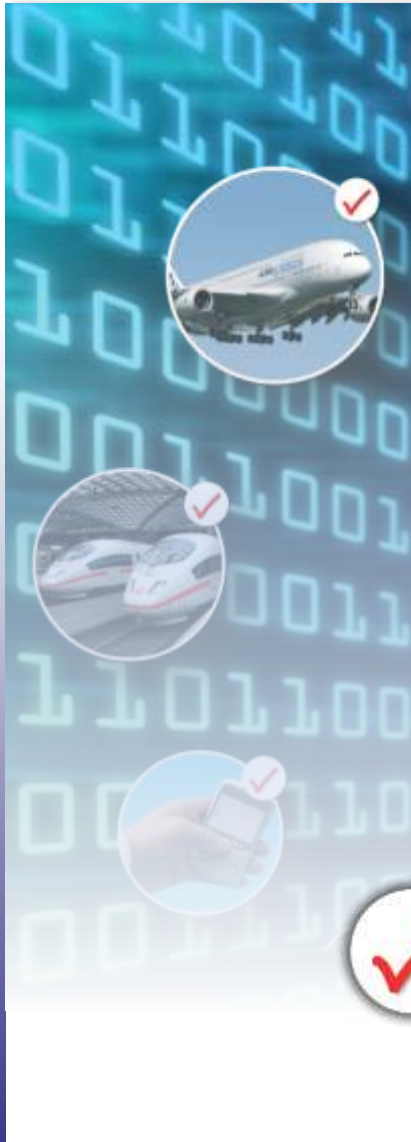


Measurement of Software Complexity with *Testwell CMT++* *Testwell CMTJava*



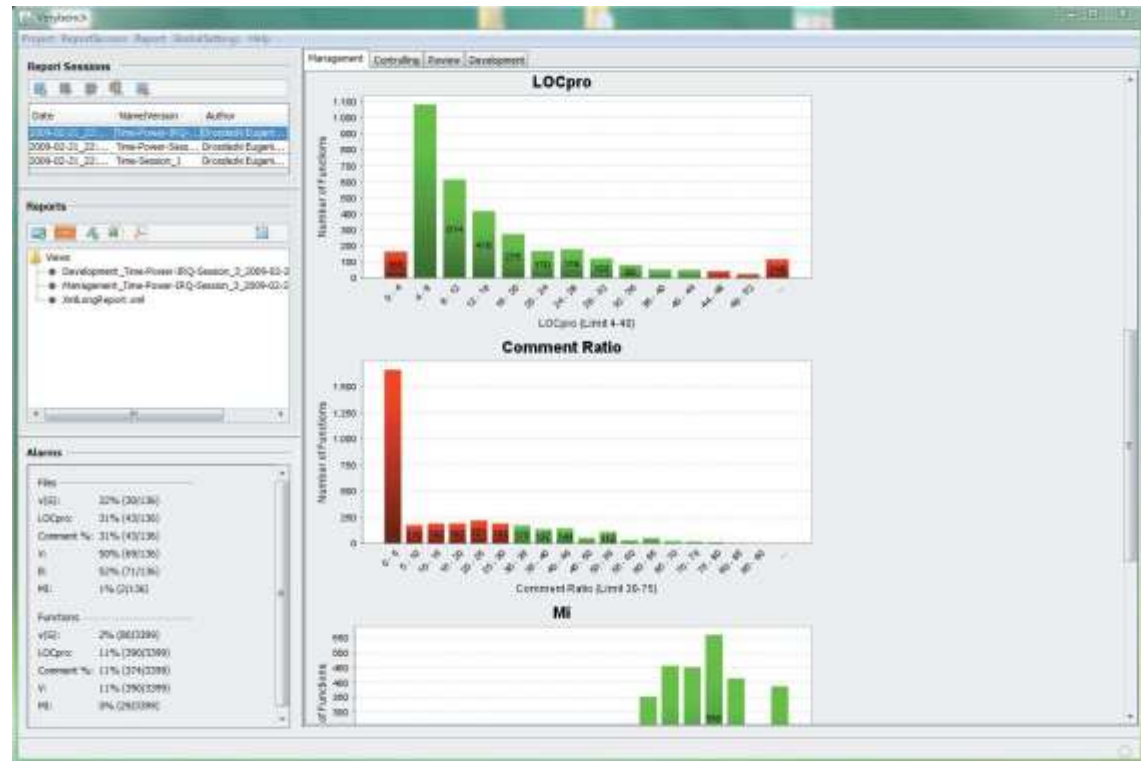


Code Complexity Measurements



Code Complexity Measurement Tools

Testwell CMT++ for C, C++ (and C#)
 Testwell CMTJava for Java

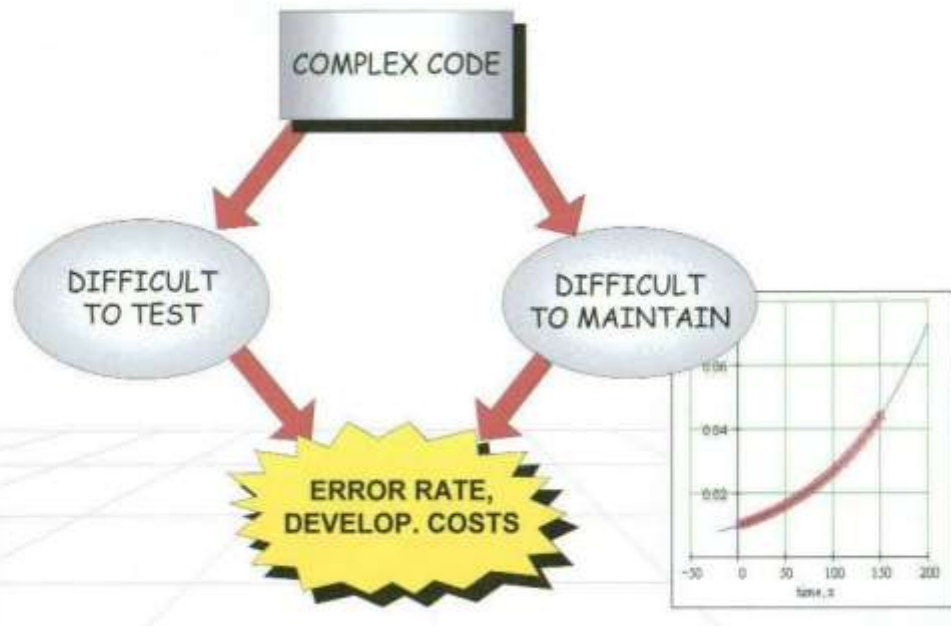




Code Complexity Measurements

Code complexity correlates with the defect rate and robustness of the application program

Wish to locate complex code



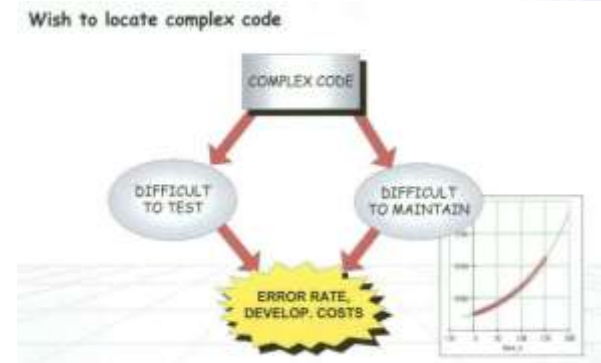


Code Complexity Measurements



Code with good complexity:

- ✓ contains less errors
- ✓ is easier and faster to test
- ✓ is easier to understand
- ✓ is easier to maintain



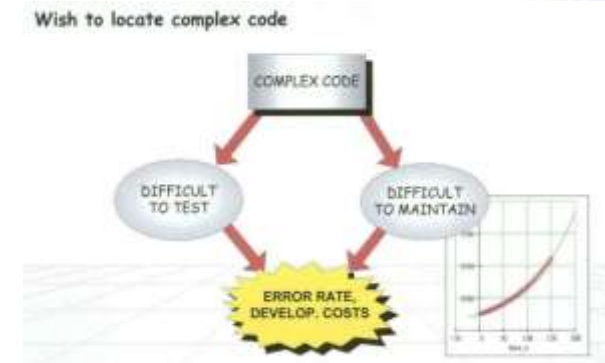


Code Complexity Measurements

Code complexity metrics are used to locate complex code

To obtain a high quality software with low cost of testing and maintenance, the code complexity should be measured as early as possible in coding.

→ developer can adapt his code when recommended values are exceeded.

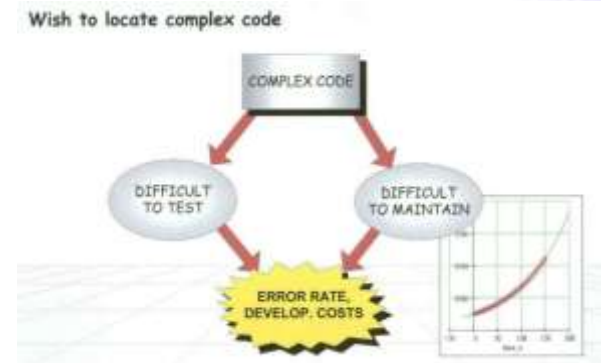




Code Complexity Measurements

Metrics shown by Testwell CMT++ / CMTJava:

- ✓ Lines of Code metrics
- ✓ McCabe Cyclomatic number
- ✓ Halstead Metrics
- ✓ Maintainability Index





Lines of code metrics





Lines of code metrics



Testwell CMT++/CMTJava calculates the following lines-of-code metrics:

- **LOCphy**: number of physical lines
- **LOCbl**: number of blank lines (a blank line inside a comment block is considered to be a comment line)
- **LOCpro**: number of program lines (declarations, definitions, directives, and code)
- **LOCcom**: number of comment lines

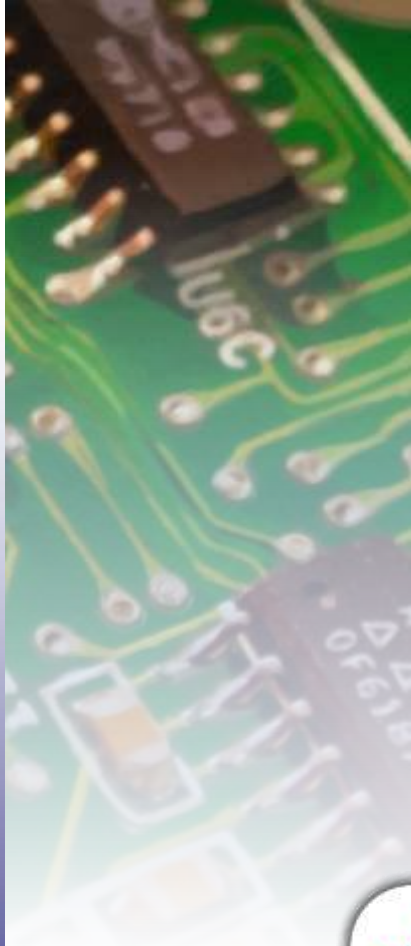
Recommendations:

Function length should be 4 to 40 program lines.

File length should be 4 to 400 program lines.

At least 30 % and at most 75 % of a file should be **comments**.

.



McCabe Cyclomatic Number



McCabe Cyclomatic Number

$v(G)$ is the number of conditional branches.
 $v(G) = 1$ for a program consisting of only sequential statements.

For a single function; $v(G)$ is one less than the number of conditional branching points in the function.

The greater the cyclomatic number is the more execution paths there are through the function, and the harder it is to understand.





Recommandations:

- The cyclomatic number of a function should be less than 15.
If a function has a cyclomatic number of 15, there are at least 15 (but probably more) execution paths through it.
- More than 15 paths are hard to identify and test.
Functions containing one selection statement with many branches make up an exception.
- A reasonable upper limit Cyclomatic number of a file is 100.





Halstead Metrics



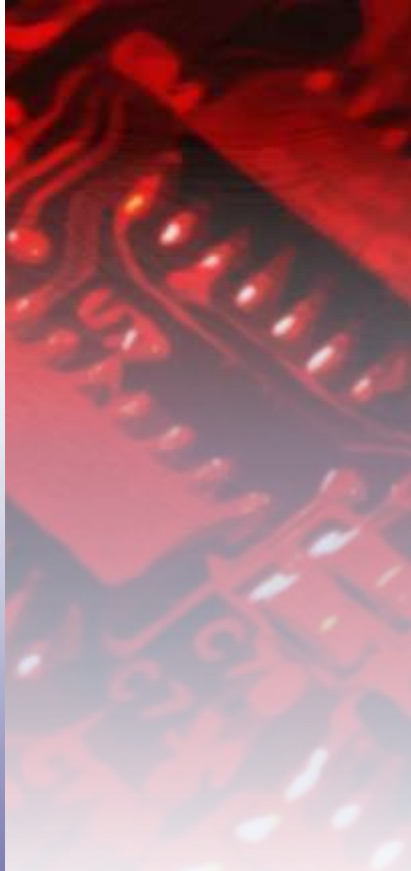


Halstead metrics



B	Estimated number of bugs
D	difficulty level, error proneness
E	effort to implement
L	program level
N	program length
N1	number of operators
N2	number of operands
n	vocabulary size (n_1+n_2)
n1	number of unique operators
n2	number of unique operands
T	implementation time / time to understand
V	volume: size of the implementation of an algorithm





B is an important metric for dynamic testing:

The number of delivered bugs approximates the number of errors in a module.

As a goal at least that many errors should be found from the module in its testing.





Maintainability Index (MI)





Maintainability Index (MI, with comments) values:

85 and more

Good maintainability

65-85

Moderate maintainability

< 65

Difficult to maintain

with really bad pieces of code (big, uncommented, unstructured) the MI value can be even negative





CMT++ Summary Report

file:///C:/Users/ed/cmt/CMHTML/index.html

Links anpassen Weitere Lesezeichen

```

.....
CMT++, Complexity Measures Tool for C/C++, Version 4.2
.....
COMPLEXITY MEASURES REPORT
.....
Copyright (c) 1993-2007 Testwell Oy
.....
License notice: This is a limited period evaluation copy license.

The input CMT++ report was produced at Mon Jan 12 17:11:08 2009
cmt options: -o C:\Users\ed\cmt\report.tmp -f C:\Users\ed\cmt\files.txt
Html'ized by cmt2html v2.2 at Mon Jan 12 17:11:08 2009
cmt2html options: -i C:\Users\ed\cmt\report.tmp

This is SUMMARY view. Go to DETAILED view. See instructions.

Alarms-1
-----
Measured object          v(G)  LOCphy  LOCpro  c%   Y   B   MI
-----
calibrate.c              20    172    107    - 3195 1.24 110
do_mounts.c              69    406    321   -11535- 3.18- 105
do_mounts_md.c           47    280    213   - 8280- 2.39- 88
do_mounts_md.c           46    429    313   -11742- 3.05- 107
msgutil.c                15    127     96   - 2751 1.02  95

OVERALL (24 %)
-----
OVERALL SUMMARY:
-----
Measure                   5 Files          40 Functions
Alarmed % Limits   Alarmed % Limits
-----
Cyclomatic number v(G)    0 0 1-100         4 10 1-15
Program lines LOCpro      0 0 4-400        12 30 4-40
Comment %                 5 100 30-75      19 47 30-75
Volume V                  3 60 100-8000    9 22 20-1000
Estimated number of bugs B 3 60 0-2         0 0 n/a
Maintainability index MI  0 0 65-          1 0 65-

```



Testwell CMT++/ CMTJava



```

report - Editor
Datei Bearbeiten Format Ansicht ?
-----
CMT++, Complexity Measures Tool for C/C++, Version 4.2
-----
COMPLEXITY MEASURES REPORT
-----
Copyright (c) 1993-2007 Testwell oy
-----

License notice: this is a limited period evaluation copy license.

This report was produced at Mon Jan 12 17:17:18 2009
Options: -o C:\Users\ed\cmt\report.txt -f C:\Users\ed\cmt\files.txt

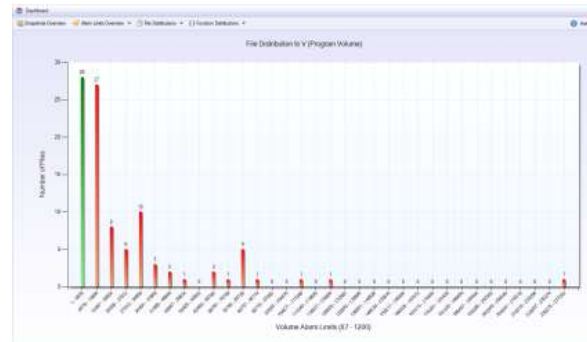
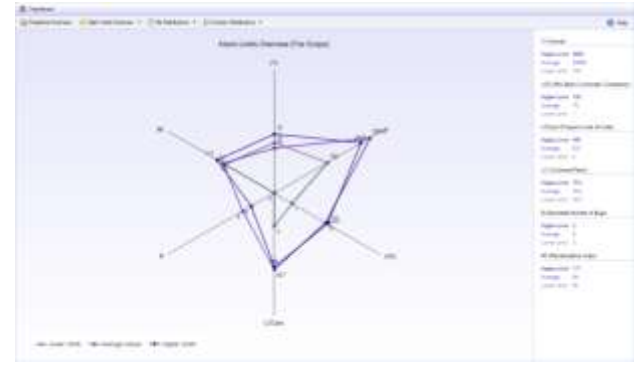
File: C:\Users\ed\Desktop\TestFiles\linux-2.6.26.8\linux-2.6.26.8\init\calibrate.c
-----
Line Measured object          v(G) LOCphy LOCpro  c%    v    B  MI
-----
13  lpi_setup()                 1     5     5     92  0.02 121
31  calibrate_delay_direct()     9    72    42~  1123- 0.39 101
104 calibrate_delay_direct()   1     1     1~   38  0.01 152
114 calibrate_delay()          10    59    47~  1146- 0.39  96
-----
172 calibrate.c                 20   172   107  - 3195  1.24 110
-----

File: C:\Users\ed\Desktop\TestFiles\linux-2.6.26.8\linux-2.6.26.8\init\do_mounts.c
-----
Line Measured object          v(G) LOCphy LOCpro  c%    v    B  MI
-----
31  load_randisk()                1     3     3    104  0.02 121
38  readonly()                    2     7     7     90  0.02 116
46  readwrite()                   2     7     7     96  0.02 115
57  name_to_dev_t()              22~   86   55~  1832- 0.57  90
144 root_dev_setup()            1     5     3     88  0.02 121
152 rootwait_setup()            2     7     7     88  0.02 116
163 root_data_setup()           1     5     3     61  0.01 123
170 fs_names_setup()           1     5     3     61  0.01 123
177 root_delay_setup()          1     5     3     92  0.02 121
187 get_fs_names()              7    26   24  -   615  0.19  83
214 do_mount_root()             3    14   13  -   520  0.12  95
229 mount_block_root()          9    52   44~  1127- 0.30  91
283 mount_nfs_root()            3    10     9    186  0.04 106
296 change_floppy()             9    27   27  -   889  0.22  82
325 mount_root()                9    28   26  -   447  0.07  98
354 prepare_namespace()        14    52   38  -   955  0.19  91
-----
406 do_mounts.c                 69   406   321  - 11535- 3.18- 105
-----

```



Verybench for Testwell CMT++/ CMTJava



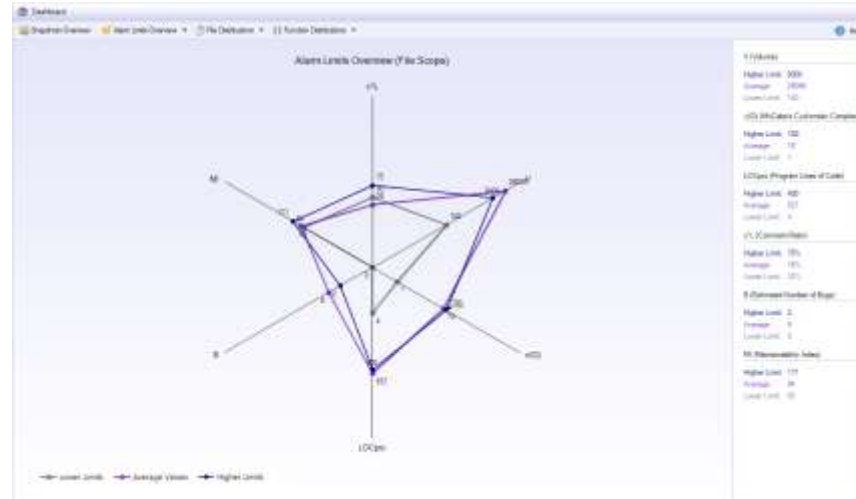
File Name	Function	Lines	LOCs	V	E	W	W
main		25	100	100	100	100	100
main2		15	2	10	20	10	10
main3		10	2	10	20	10	10
main4		10	2	10	20	10	10
main5		10	2	10	20	10	10
main6		10	2	10	20	10	10

Verybench:
Graphical front-end for Testwell Complexity Measurement Tools



Snapshots Overview

shows the course of the measured source code's quality over time by stating the alarm ratios of the latest six snapshots.

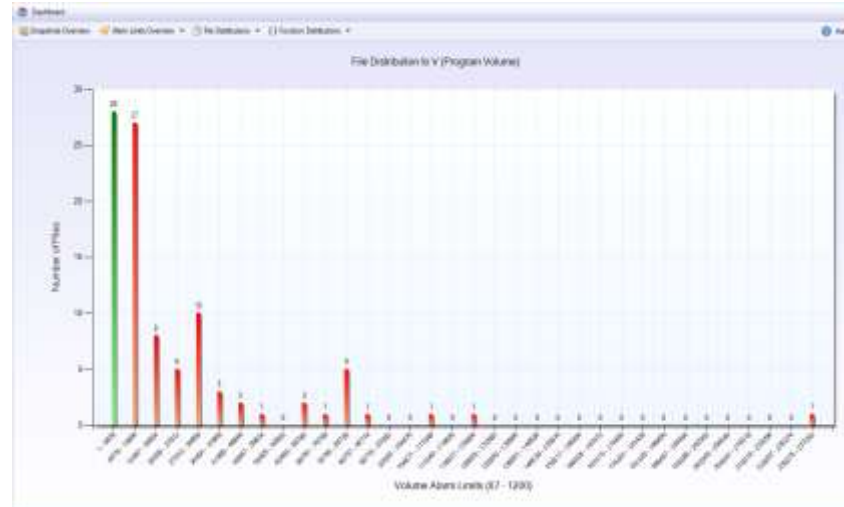


Alarm-Limits Overview

integrates all configurable Alarm Limits into a Radar Chart for each file and function.

Every axis represents a different alarming metric with its configurable lower and higher Alarm Limits.

The Radar Chart basically shows the deviation of a metric's current value from its lower and higher Alarm Limits.



Distribution of Metrics

shown on file and function levels for:

V (Program Volume)

c% (Comment Ratio)

LOCpro (Program Lines of Code)

v(G) (McCabe' Cyclomatic Number)

In addition B (Estimated Number of Bugs) is shown on file level



Metrics

File Scope Metrics (Snapshot: 6 Files) / Max. Alarms: 36

File Name	Functions	Alarms	a%	c%	LOCs	V	B	+SD	SB
bk-well.c	3	2	67%	33%	151	3462.76	3.882	36	127
bk-epfl.c	23	2	9%		329	1000.00	2.34	26	106
bk-leg.c	13	2	15%	38%	204	1000.00	2.34	25	122
beg.c	32	4	13%		289	1000.00	8.425	39	99
displayed.c	34	5	15%		344	1000.00	10.807	48	113
prodfile.c	11	5	45%		105	1000.00	2.34	23	112

Source Code

```

1
2 * Functions related to safety, to completion
3
4 #include <sys/types.h>
5 #include <sys/time.h>
6 #include <sys/timeb.h>
7 #include <sys/stat.h>
8 #include <sys/param.h>
9 #include <sys/errno.h>
10 #include <sys/unistd.h>
11
12 #include "bk.h"
13
14 #define DEFINE_PER_OP_LISTEN fd_head, bk_cpu_bond
15
16
17 * Setting action handler - move entries to local list and keep over them
18 * while passing them to the alarm registered handler
19
20 #define void bk_alarm_getting(fd_at, action "O")
21
22 #define fd_head "cpu_jar, local_jar"

```

Metrics View

overview over all (both alarming and non-alarming) metrics of the measured files and functions



Thank You



Thank you for your time!

Your Verifysoft Team