



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

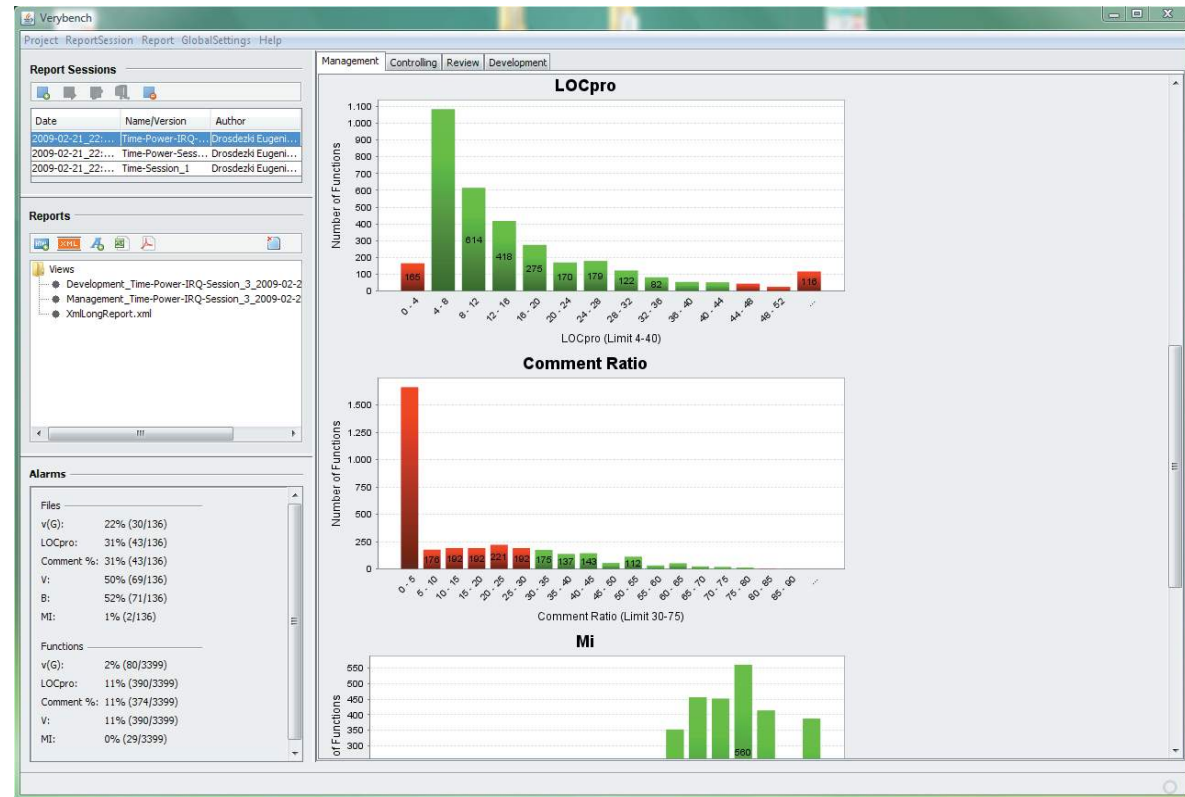
© 2011 Verifysoft Technology GmbH

07.01.2011


TECHNOLOGY

Testwell CMT++ / CMTJava

Testwell CMT++
Testwell CMTJava
Code Complexity
Measures Tools
for C/C++ and Java



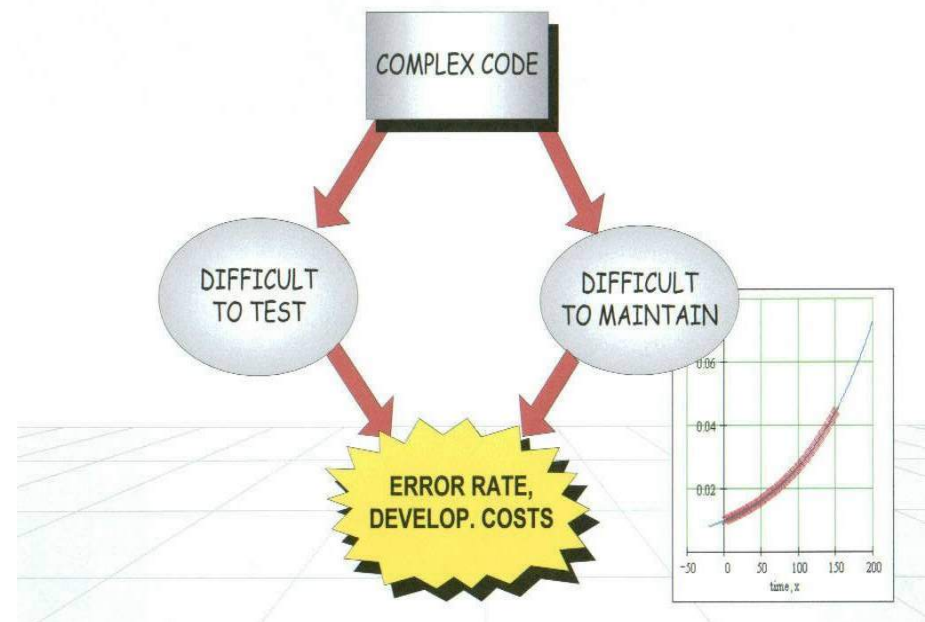
Testwell CMT++ / CMTJava

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

Complex code

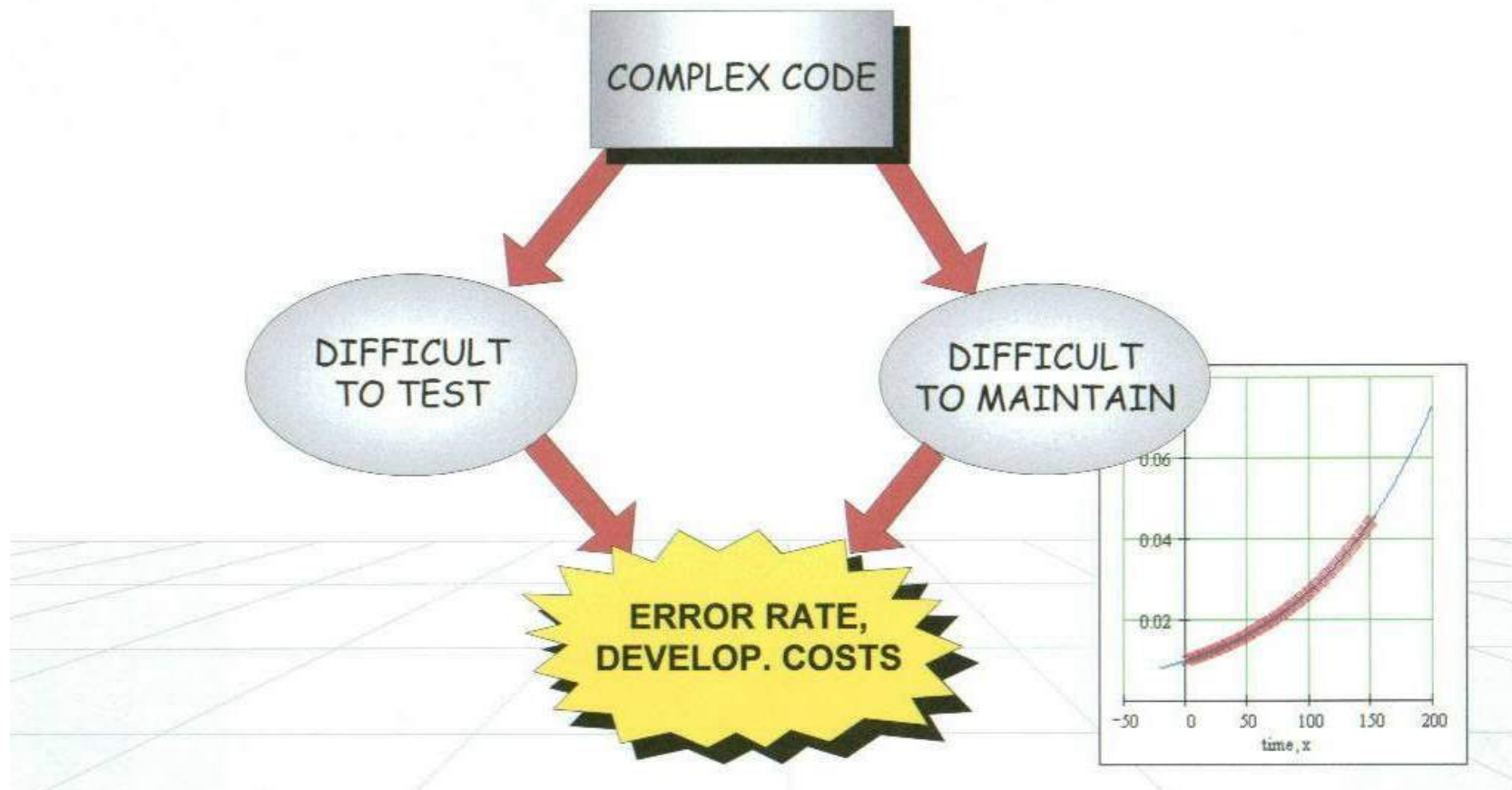
- contains more errors
- is difficult to test
 - less errors are found during testing
- is difficult to maintain

Wish to locate complex code



Testwell CMT++ / CMTJava

Wish to locate complex code





Testwell CMT++ / CMTJava

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.



Testwell CMT++ / CMTJava

Metrics shown by Testwell CMT++ / CMTJava

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



Lines-of-Code Metrics



Testwell CMT++ / CMTJava

Most traditional measures used to quantify software complexity. They are simple, easy to count, and very easy to understand. They do not, however, take into account the intelligence content and the layout of the code.

Testwell CMT++ 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

Testwell CMT++ / CMTJava

Recommendations:

Function length should be 4 to 40 program lines.

- A function definition contains at least a prototype, one line of code, and a pair of braces, which makes 4 lines.
- A function longer than 40 program lines probably implements many functions. (*Exception: Functions containing one selection statement with many branches*)
- Decomposing them into smaller functions often decreases readability.



Testwell CMT++ / CMTJava

Recommendations:

File length should be 4 to 400 program lines.

- The smallest entity that may reasonably occupy a whole source file is a function, and the minimum length of a function is 4 lines.
- Files longer than 400 program lines (10..40 functions) are usually too long to be understood as a whole.

Testwell CMT++ / CMTJava

Recommendations:

Comments

- At least 30 % and at most 75 % of a file should be comments.
- If less than one third of a file is comments the file is either very trivial or poorly explained.
- If more than 75% of a file are comments, the file is not a program but a document.

(Exeption: In a well-documented header file percentage of comments may sometimes exceed 75%)



McCabe Cyclomatic Number : $v(G)$



Testwell CMT++ / CMTJava

The cyclomatic complexity $v(G)$ has been introduced by Thomas McCabe in 1976.

Measures the number of linearly-independent paths through a program module (Control Flow).

The McCabe complexity is one of the more widely-accepted software metrics, it is intended to be independent of language and language format.

Considered as a broad measure of soundness and confidence for a program.



Testwell CMT++ / CMTJava

$v(G)$ 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.



Testwell CMT++ / CMTJava

How McCabe Metrics are calculated with CMT++

Increase of McCabe cyclomatic number $v(G)$ by one:

- if-statement (introduces a new branch to the program)
- Iteration constructs such as for- and while-loops
- Each case ...: part in the switch-statement
- Each catch (...) part in a try-block
- Construction `expr1 ? expr2 : expr3`.



Testwell CMT++ / CMTJava

In CMT++ the branches generated by conditional compilation directives are also counted to $v(G)$.

(Even if conditional compilation directives do not add branches to the control flow of the executable program, they increase the complexity of the program file that the user sees and edits.)

$v(G)$ is insensitive to unconditional branches like goto-, return- and break-statements although they surely increase complexity.



Testwell CMT++ / CMTJava

- In summary, the following language constructs increase the cyclomatic number by one: *if (...)*, *for (...)*, *while (...)*, *case*; *catch (...)*, *&&*, *||*, *?*, *#if*, *#ifdef*, *#ifndef*, *#elif*.



Testwell CMT++ / CMTJava

For dynamic testing is concerned, the cyclomatic number $v(G)$ is one of the most important complexity measures.

Because the cyclomatic number describes the control flow complexity, it is obvious that modules and functions having high cyclomatic number need more test cases than modules having a lower cyclomatic number.

Rule: each function should have at least as many test cases as indicated by its cyclomatic number.

Testwell CMT++ / CMTJava

Recommendations:

- 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



Testwell CMT++ / CMTJava

- Developed by Maurice Halstead (sen.)
- Introduced 1977
- Used and experimented extensively since that time

They are one of the oldest measures of program complexity

- Strong indicators of code complexity.
- Often used as a maintenance metric.



Testwell CMT++ / CMTJava

Halstead's metrics is based on interpreting the source code as a sequence of tokens and classifying each token to be an operator or an operand.

Then is counted

- number of unique (distinct) operators ($n1$)
- number of unique (distinct) operands ($n2$)
- total number of operators ($N1$)
- total number of operands ($N2$)

All other Halstead measures are derived from these four quantities with certain fixed formulas as described later.

Testwell CMT++ / CMTJava

Operands:

IDENTIFIER	all identifiers that are not reserved words
TYPENAME	
TYPESPEC	(type specifiers) Reserved words that specify type: <i>bool, char, double, float, int, long, short, signed, unsigned, void</i> . This class also includes some compiler specific nonstandard keywords.
CONSTANT	Character, numeric or string constants.

Testwell CMT++ / CMTJava

Operators:

SCSPEC

(storage class specifiers) Reserved words that specify storage class: *auto*, *extern*, *inline*, *register*, *static*, *typedef*, *virtual*, *mutable*.

TYPE_QUAL

(type qualifiers) Reserved words that qualify type: *const*, *friend*, *volatile*.

RESERVED

Other reserved words of C++: *asm*, *break*, *case*, *class*, *continue*, *default*, *delete*, *do*, *else*, *enum*, *for*, *goto*, *if*, *new*, *operator*, *private*, *protected*, *public*, *return*, *sizeof*, *struct*, *switch*, *this*, *union*, *while*, *namespace*, *using*, *try*, *catch*, *throw*, *const_cast*, *static_cast*, *dynamic_cast*, *reinterpret_cast*, *typeid*, *template*, *explicit*, *true*, *false*, *typename*. This class also includes some compiler specific nonstandard keywords.

OPERATOR

One of the following: ! != % %= & && || &= () * *= + ++ += , - -- -= -> / /= : :: < << <<= <= = == > >= >> >>= ? [] ^ ^= { } | |= ~ts.



Testwell CMT++ / CMTJava

The following control structures *case ...: for (...) if (...)* *seitch (...)* *while for (...)* and *catch (...)* are treated in a special way.

The colon and the parentheses are considered to be a part of the constructs.

The *case* and the colon or the *for (...)* *if (...)* *switch (...)* *while for (...)* and *catch (...)* and the parentheses are counted together as one operator.



Testwell CMT++ / CMTJava

Program length (N)

The program length (N) is the sum of the total number of operators and operands in the program:

$$N = N1 + N2$$

Vocabulary size (n)

The vocabulary size (n) is the sum of the number of unique operators and operands:

$$n = n1 + n2$$



Testwell CMT++ / CMTJava

Program volume (V)

= information content of the program

It is calculated as the program length times the 2-base logarithm of the vocabulary size (n):

$$V = N * \log_2(n)$$

Halstead's volume (V) describes the size of the implementation of an algorithm.



Testwell CMT++ / CMTJava

Recommendations:

- The volume of a **function** should be at least 20 and at most 1000.
- The volume of a parameterless one-line function that is not empty; is about 20.
- A volume greater than 1000 tells that the function probably does too many things.

The volume of a **file** should be at least 100 and at most 8000.

These limits are based on volumes measured for files whose LOCpro and v(G) are near their recommended limits.

Testwell CMT++ / CMTJava

Difficulty level (D)

The difficulty level or error proneness (D) of the program is proportional to the number of unique operators in the program.

D is also proportional to the ration between the total number of operands and the number of unique operands.

(i.e. if the same operands are used many times in the program, it is more prone to errors)

$$D = (n1 / 2) * (N2 / n2)$$



Testwell CMT++ / CMTJava

Program level (L)

The program level (L) is the inverse of the error proneness of the program.

I.e. A low level program is more prone to errors than a high level program.

$$L = 1 / D$$



Testwell CMT++ / CMTJava

Effort to implement (E)

The effort to implement (E) or understand a program is proportional to the volume and to the difficulty level of the program.

$$E = V * D$$



Testwell CMT++ / CMTJava

Time to implement (T)

The time to implement or understand a program (T) is proportional to the effort.

Halstead has found that dividing the effort by 18 give an approximation for the time in seconds.

$$T = E / 18$$

Testwell CMT++ / CMTJava

Number of delivered bugs (B)

The number of delivered bugs (B) correlates with the overall complexity of the software.

$$B = \frac{E^{2/3}}{3000}$$

Estimate for the number of errors in the implementation.
Delivered bugs in a file should be less than 2.

Experiences have shown that, when programming with C or C++, a source file almost always contains more errors than B suggests.



Testwell CMT++ / CMTJava

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)



Testwell CMT++ / CMTJava

Calculated with certain formulae from lines-of-code measures, McCabe measure and Halstead measures.

Indicates when it becomes cheaper and/or less risky to rewrite the code instead to change it.



Testwell CMT++ / CMTJava

Two variants of Maintainability Index:

- One that contains comments (MI) and one that does not contain comments (MIwoc).

Actually there are three measures:

- MIwoc: Maintainability Index without comments
- MIcw: Maintainability Index comment weight
- MI: Maintainability Index = MIwoc + MIcw

Testwell CMT++ / CMTJava

$$Mlwoc = 171 - 5.2 * \ln(\text{aveV}) - 0.23 * \text{aveG} - 16.2 * \ln(\text{aveLOC})$$

- aveV = average Halstead Volume V per module
- aveG = average extended cyclomatic complexity $v(G)$ per module
- aveLOC = average count of lines LOC_{phy} per module
- perCM = average percent of lines of comments per Module

Testwell CMT++ / CMTJava

$$Mlcw = 50 * \sin(\sqrt{2,4 * perCM})$$

- perCM = average percent of lines of comments per module

$$MI = Mlwoc + Mlcw$$

Testwell CMT++ / CMTJava

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



Testwell CMT++ / CMTJava

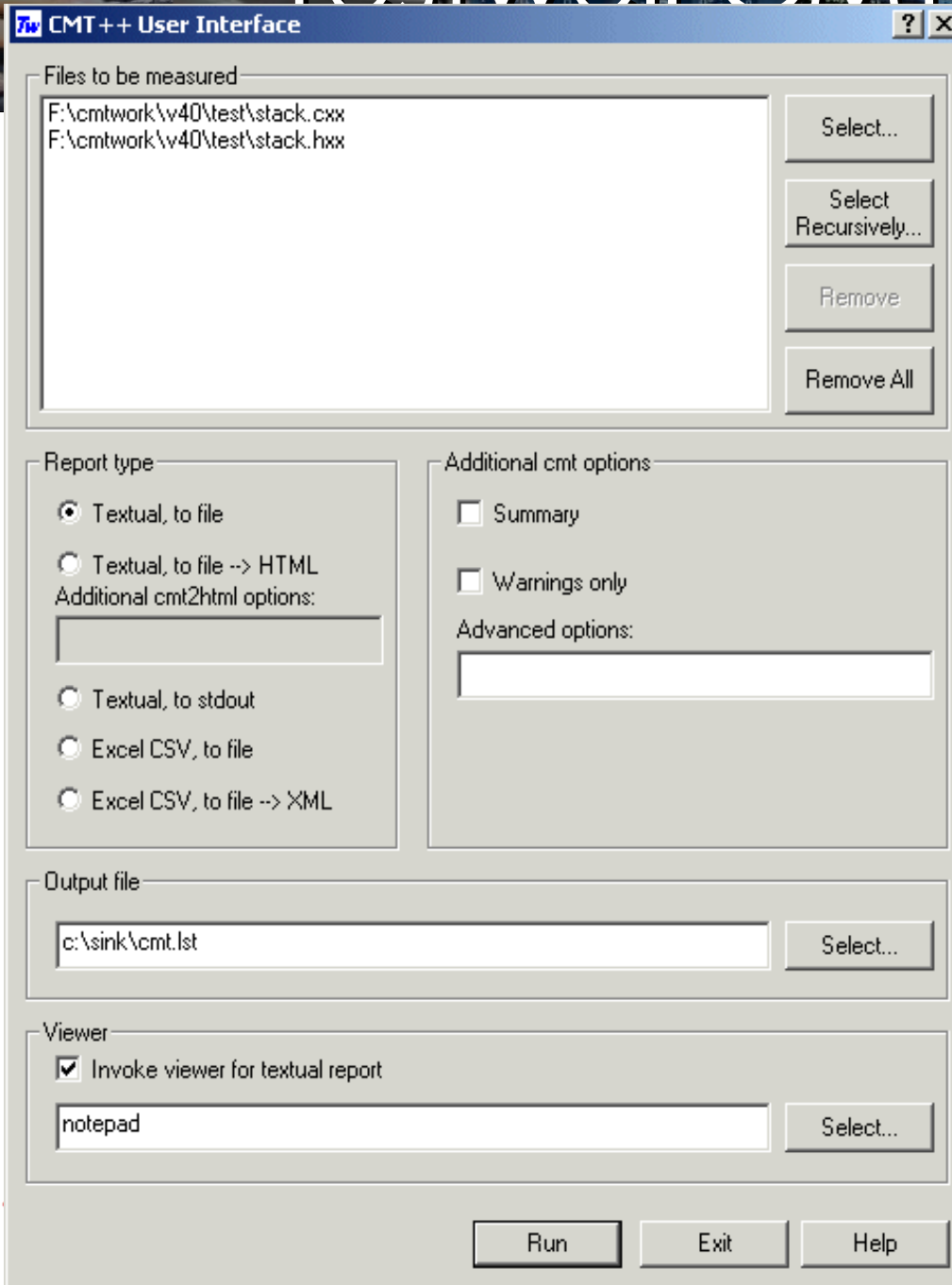
Testwell CMT++ / CMTJava User Interfaces:

„native“ interface

Verybench

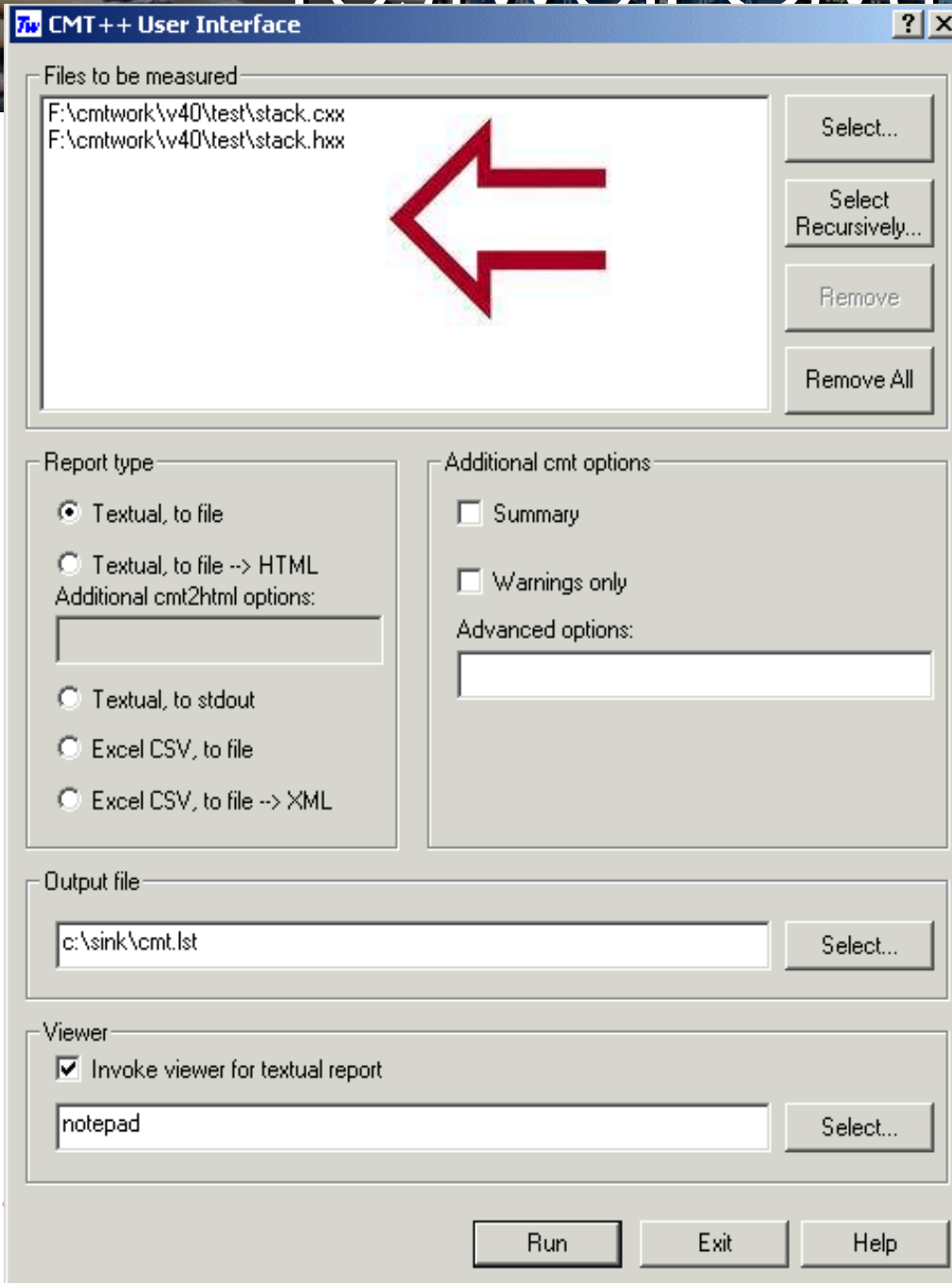
(graphical frontend)

Testwell CMT++ / CMTJava



GUI for MS Windows

Testwell CMT++ / CMTJava



Selecting files for measurement

Selection of one file, several or (recursively) all files located in a directory and its subdirectories.

If you have selected several files and highlighted one of them, the measurement will be performed only for this file.

Testwell CMT++ / CMTJava

CMT++ User Interface

Files to be measured

F:\cmtwork\w40\test\stack.cxx
F:\cmtwork\w40\test\stack.hxx

Select...
Select Recursively...
Remove
Remove All

Report type

Textual, to file
 Textual, to file --> HTML
Additional cmt2html options:

Additional cmt options
Additional cmt options:

Textual, to stdout
 Excel CSV, to file
 Excel CSV, to file --> XML

Output file

c:\sink\cmt.lst

Select...

Viewer

Invoke viewer for textual report

notepad

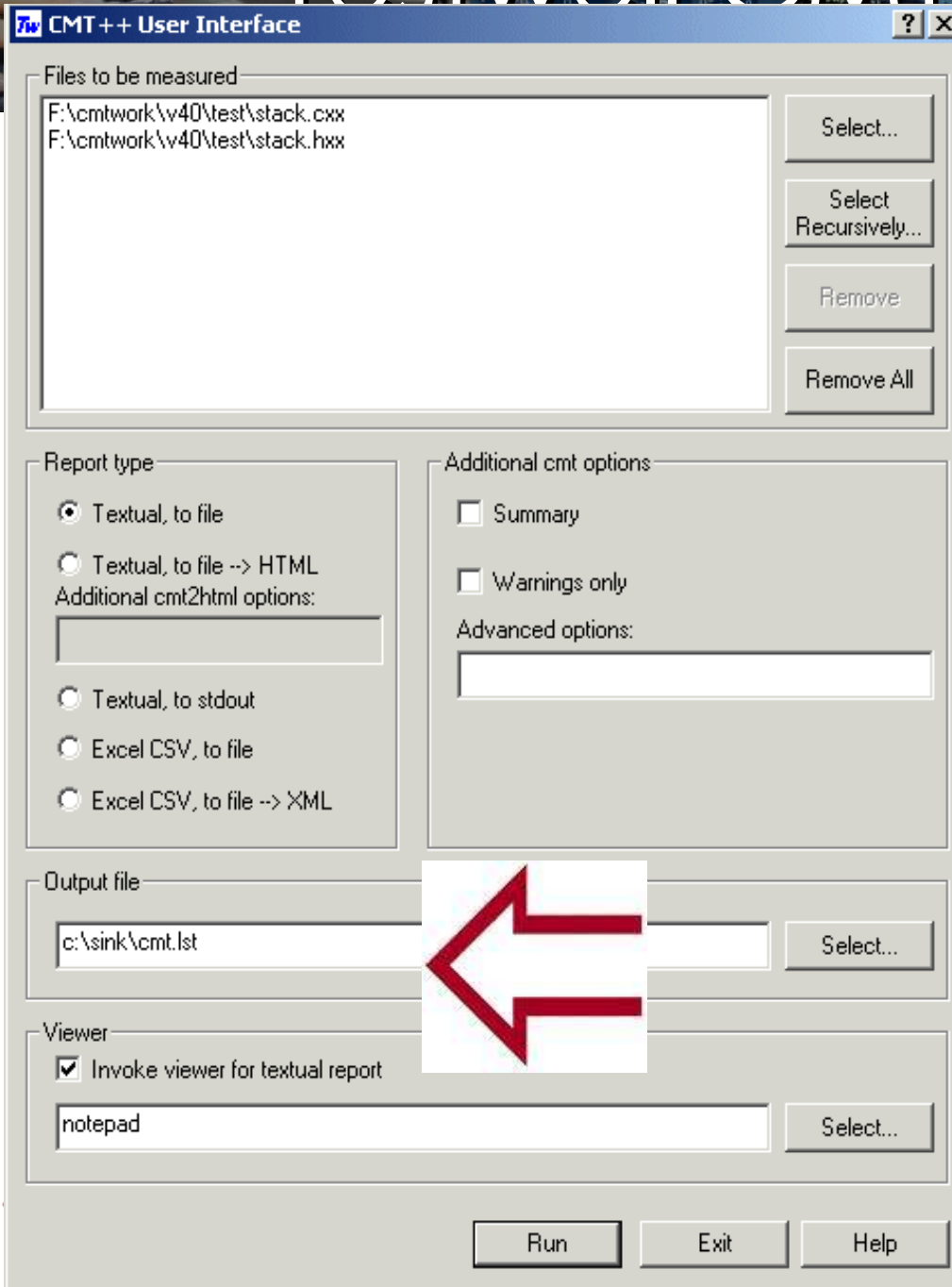
Select...

Run Exit Help

Report type

the selected report will be shown immediately after measurement, if you have a corresponding viewer installed on you computer.

Testwell CMT++ / CMTJava



Output file

you can set location and name of file with textual report, another reports will be saved in near locations, there are same default settings for locations and names of report files.



Testwell CMT++ / CMTJava

Output formats:

Short report

File level summary

HTML-report

Long report in XML

Excel report

Testwell CMT++ / CMTJava

Output formats:

Short report

File level summary

HTML-report

Long report in XML

Excel report

File: stack.h

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
60	stack.h	2	60	22		343	0.09	118

File: stack.cpp

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
32	Stack::							
32	Stack()	1	18	6		56	0.01	150
51	~Stack()	1	12	4		27	0.01	160
64	clear()	1	12	4		33	0.01	159
77	height()	1	12	4		29	0.01	159
90	push()	3	24	14		288	0.10	130
115	pop()	2	18	7		94	0.03	141
134	top()	2	17	7		100	0.03	143
152	element()	1	1	1-		52	0.01	150
152	stack.cpp	5	152	49		972	0.41	143

File: demofile.h

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
8	MyClass::							
8	MyClass()	1	3	3-		20-	0.00	137
11	~MyClass()	1	3	2		20-	0.00	137

Testwell CMT++ / CMTJava

Output formats:

Options: -s -o report-s.txt

Short report

Line	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
60	stack.h	2	60	22		343	0.09	118
152	stack.cpp	5	152	49		972	0.41	143
18	demofile.h	1	18	12	-	156	0.04	142
41	demofile.cpp	11	41	33	-	620	0.22	120

File level summary

OVERALL SUMMARY:

HTML-report

Long report in XML

Excel report

Measure	4 Files			0 Functions		
	Alarmed	%	Limits	Alarmed	%	Limits
Cyclomatic number v(G)	0	0	1-100	0	0	1-15
Program lines LOCpro	0	0	4-400	0	0	4-40
Comment %	2	50	30-75	0	0	30-75
Volume V	0	0	100-8000	0	0	20-1000
Estimated number of bugs B	0	0	0-2	0	0	n/a
Maintainability index MI	0	0	65-	0	0	65-
Total	2	8		0	0	

Files: 4 LOCphy: 271 LOCbl: 45 LOCpro: 116 LOCcom: 113 ';': 51
v(G) : 16 MI without comments : 107 MI comment weight : 40 MI: 147

Testwell CMT++ / CMTJava

Output formats:

Short report

File level summary

HTML-report

Long report in XML

Excel report

This is SUMMARY view. Go to [DETAILED](#) view. See [instructions](#).

Alarms-%	Measured object	v(G)	LOCphy	LOCpro	c%	V	B	MI
<div style="width: 20px; height: 10px; border: 1px solid black;"></div>	stack.h	2	60	22		343	0.09	118
<div style="width: 20px; height: 10px; border: 1px solid black; background-color: #ccc;"></div>	stack.cpp	5	152	49		972	0.41	143
<div style="width: 20px; height: 10px; border: 1px solid black; background-color: #f00;"></div>	demofile.h	1	18	12	-	156	0.04	142
<div style="width: 20px; height: 10px; border: 1px solid black; background-color: #f00;"></div>	demofile.cpp	11	41	33	-	620	0.22	120
<div style="width: 20px; height: 10px; border: 1px solid black; background-color: #f00;"></div>	OVERALL (8 %)							

OVERALL SUMMARY:

Measure	4 Files			13 Functions		
	Alarmed	%	Limits	Alarmed	%	Limits
Cyclomatic number v(G)	0	0	1-100	0	0	1-15
Program lines LOCpro	0	0	4-400	2	15	4-40
Comment %	2	50	30-75	1	7	30-75
Volume V	0	0	100-8000	2	15	20-1000
Estimated number of bugs B	0	0	0-2	0	0	n/a
Maintainability index MI	0	0	65-	0	0	65-
Total	2	8		5	7	

Files: 4 LOCphy: 271 LOCbl: 45 LOCpro: 116 LOCcom: 113 '': 51
v(G) : 16 MI without comments : 107 MI comment weight : 40 MI: 147

Testwell CMT++ / CMTJava

Output formats:

Short report

File level summary

HTML-report

Long report in XML

Excel report

```
<file name="stack.cpp">
  <function name="Stack::Stack()">
    <start_line>32</start_line>
    <vG>1</vG>
    <LOCphy>18</LOCphy>
    <LOCpro>6</LOCpro>
    <LOCb1>1</LOCb1>
    <LOCcom>11</LOCcom>
    <N>17</N>
    <N1>9</N1>
    <N2>8</N2>
    <n>10</n>
    <n1>5</n1>
    <n2>5</n2>
    <V>56.473</V>
    <B>0.012</B>
    <D>4.000</D>
    <E>225.891</E>
    <L>0.250</L>
    <T>00:00:12</T>
    <MaxND>1</MaxND>
    <MIwoc>103</MIwoc>
    <MIcwc>47</MIcwc>
    <MI>150</MI>
  </function>
  <function name="Stack::~~Stack()">
    <start_line>51</start_line>
```

Testwell CMT++ / CMTJava

Output formats:

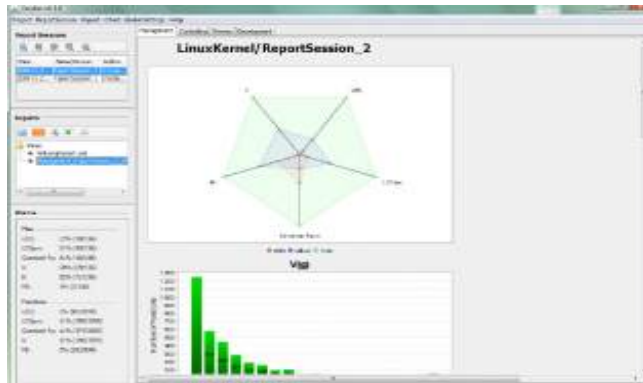
Short report

File level summary

	File	Line	Measured object v(G)	MaxND	LOCphy	LOCbl	LOCpro	LOCcom	V	B (x100)	T
HTML-report	"stack.cpp"	32	"Stack::Stack()"		1	1	18	1	6	11	56
	"stack.cpp"	51	"Stack::~Stack()"		1	1	12	1	4	7	27
	"stack.cpp"	64	"Stack::clear()"		1	1	12	1	4	7	33
Long report in	"stack.cpp"	77	"Stack::height()"		1	1	12	1	4	7	29
	"stack.cpp"	90	"Stack::push()" 3		2	24	1	14	9	288	10
	"stack.cpp"	115	"Stack::pop()" 2		1	18	4	7	7	94	3
Excel report	"stack.cpp"	134	"Stack::top()" 2		1	17	3	7	7	100	3
	"stack.cpp"	152	"Stack::element()"		1	1	1	0	1	0	52
	"demofile.h"	8	"MyClass::MyClass()"		1	1	3	0	3	0	20
	"demofile.h"	11	"MyClass::~MyClass()"		1	1	3	0	2	0	20
	"demofile.cpp"	9	"MyClass::foo1()"		6	2	6	0	6	0	164
	"demofile.cpp"	16	"MyClass::foo2()"		2	2	8	0	8	0	113
"demofile.cpp"	25	"SomeFunction()"		5	3	17	0	17	0	209	

Testwell CMT++ / CMTJava

Verybench (graphical frontend):



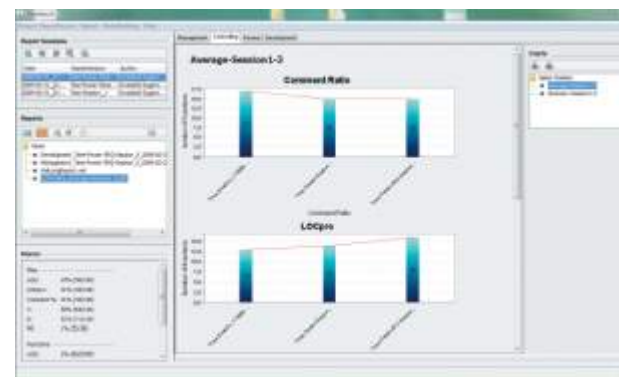
Management View



Development View



Review View



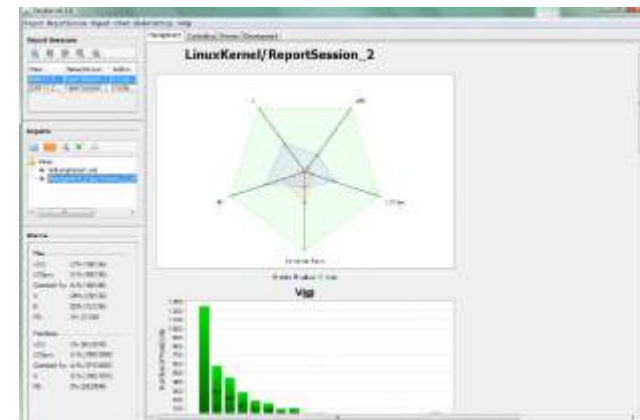
Controlling View

Testwell CMT++ / CMTJava

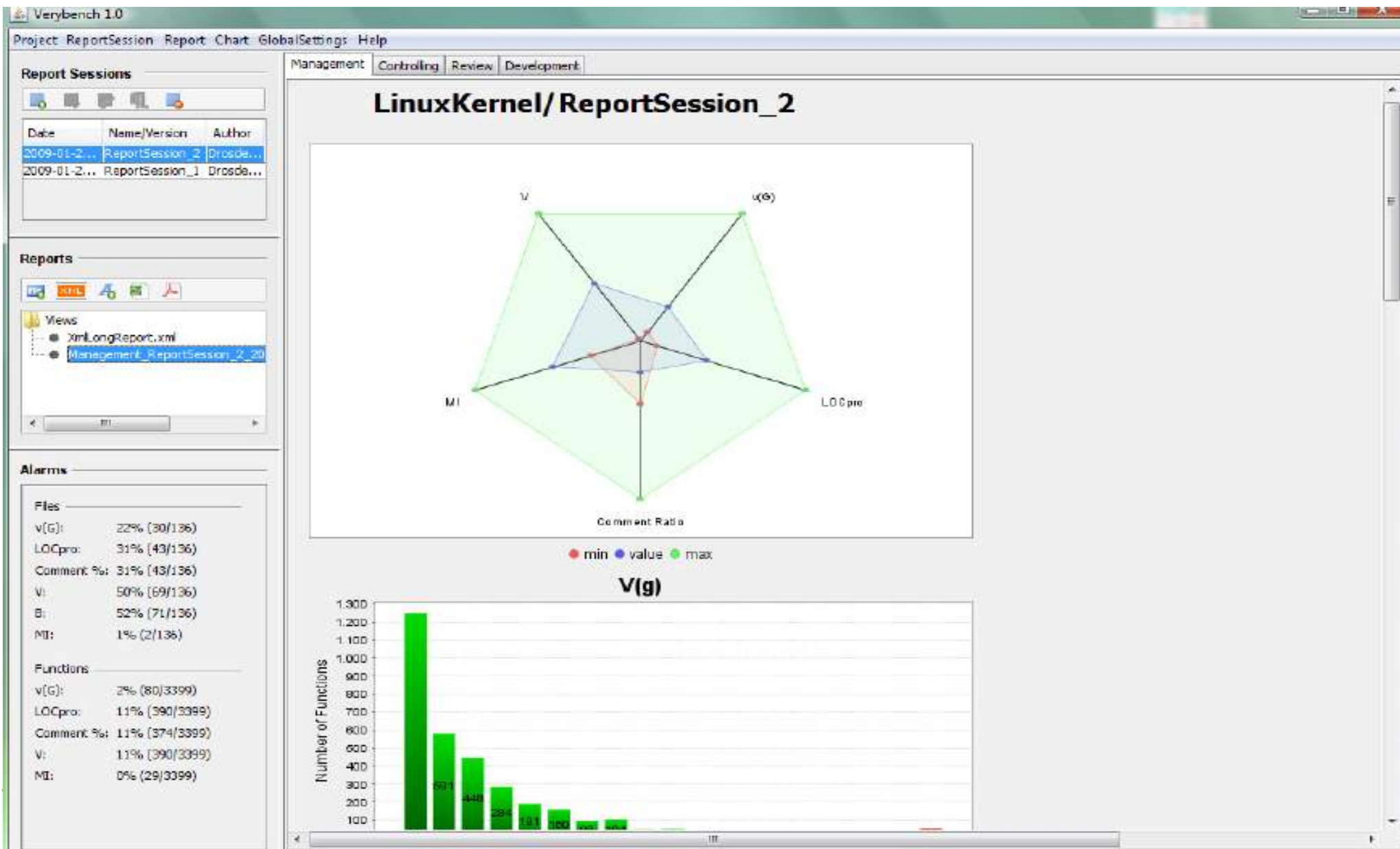
Management View

Graphically presentation of metrics

Kiviati diagramm gives a short glance about the code complexity
(ideal target values are represented by the outline of the diagram)



Testwell CMT++ / CMTJava



Testwell CMT++ / CMTJava

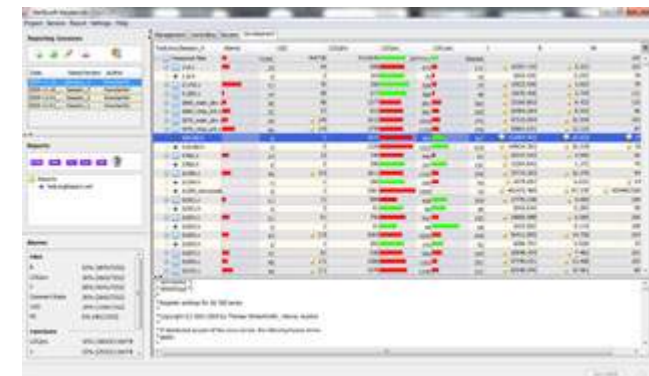
Development View

shows all files and functions of the current reporting session and the corresponding metrics

By clicking a listed file in the tree table all its functions and its source code is shown.

- * Alarms: Amount of measured values exceeding limits
- * LOCpro: Number of program lines
- * LOCcom: Number of lines with comments
- * LOCphy: Number of physical lines
- * v(G): Cyclomatic number
- * V: Program volume
- * B: Estimated number of programming errors

* MI: Maintainability Index



File	Function	LOC	LOCcom	LOCphy	v(G)	V	B	MI
...

Testwell CMT++ / CMTJava

Verifysoft Verybench

Project Session Report Settings Help

Reporting Sessions

Date	Name/Version	Author
2009-11-16 ...	Session_4	Konstantin
2009-11-16 ...	Session_3	Konstantin
2009-11-01 ...	Session_2	Konstantin
2009-11-01 ...	Session_1	Konstantin

Reports

HTML XML TXT XLS PDF

Reports

- XmlLongReport.xml

Alarms

Files

B	53% (3870/7252)
LOCpro	36% (2665/7252)
V	58% (4241/7252)
Comment Ratio	36% (2665/7252)
v(G)	18% (1358/7252)
MI	6% (481/7252)

Functions

LOCpro	16% (18624/110074)
V	22% (25252/110074)

Management Controlling Review Development

TestLinux/Session_4	Alarms	v(G)	LOCphy	LOCpro	LOCcom	V	B	MI
Measured files		72381	465728	5310836	3777711	996065		105
11d.c		23	54	698	473	111	19397.143	102
11d.h		0	2	104	70	10	1842.435	79
21142.c		11	70	258	208	37	10923.936	79
21285.c		14	58	517	386	48	15676.426	111
2860_main_dev.c		30	98	1377	801	362	33169.863	110
2860_rtmp_init.c		22	57	922	581	200	28464.264	96
2870_main_dev.c		39	140	1612	1018	372	47315.004	103
2870_rtmp_init.c		46	143	1778	1133	379	55891.031	87
300vtbl.h		0	1	1072	956	367	112587.422	37
310vtbl.h		0	5	1339	1222	618	149624.391	35
3780i.c		19	22	740	486	81	20047.533	95
3780i.h		0	2	358	247	152	10204.841	70
3c359.c		49	153	1811	1143	370	70714.203	94
3c359.h		0	1	288	200	55	8079.897	64
3c359_microcode		0	1	1581	1558	19	401473.469	4294967290
3c501.c		11	71	899	428	370	17770.238	109
3c501.h		0	2	91	58	35	2043.634	99
3c503.c		21	91	756	542	152	24685.988	100
3c503.h		0	1	91	44	68	1623.002	108
3c505.c		43	215	1665	1028	439	56411.895	104
3c505.h		0	1	292	193	92	5098.757	73
3c507.c		22	81	938	586	265	29948.424	101
3c509.c		48	171	1586	1162	251	57740.031	103
3c515.c		44	211	1579	1190	311	65548.945	89

```

/* $XFree86$ */
/* $XdotOrg$ */
/*
 * Register settings for SIS 300 series
 *
 * Copyright (C) 2001-2005 by Thomas Winischhofer, Vienna, Austria
 *
 * If distributed as part of the Linux kernel, the following license terms
 * apply:

```

show details

Testwell CMT++ / CMTJava

Review View

Similar to Development View

But shows files that exceeded
a set alarm value by one or more
metrics



File Name	Metric 1	Metric 2	Metric 3	Metric 4	Metric 5	Metric 6	Metric 7	Metric 8	Metric 9	Metric 10
File 1	100	200	300	400	500	600	700	800	900	1000
File 2	150	250	350	450	550	650	750	850	950	1050
File 3	200	300	400	500	600	700	800	900	1000	1100
File 4	250	350	450	550	650	750	850	950	1050	1150
File 5	300	400	500	600	700	800	900	1000	1100	1200
File 6	350	450	550	650	750	850	950	1050	1150	1250
File 7	400	500	600	700	800	900	1000	1100	1200	1300
File 8	450	550	650	750	850	950	1050	1150	1250	1350
File 9	500	600	700	800	900	1000	1100	1200	1300	1400
File 10	550	650	750	850	950	1050	1150	1250	1350	1450

Testwell CMT++ / CMTJava

Verifysoft Verybench

Project Session Report Settings Help

Reporting Sessions

Date	Name/Version	Author
2009-11-16_...	Session_4	Konstantin
2009-11-16_...	Session_3	Konstantin
2009-11-01_...	Session_2	Konstantin
2009-11-01_...	Session_1	Konstantin

Reports

HTML XML TXT XLS PDF

Reports

- XmlLongReport.xml

Alarms

Files

B	53% (3870/7252)
LOCpro	36% (2665/7252)
V	58% (4241/7252)
Comment Ratio	36% (2665/7252)
v(G)	18% (1358/7252)
MI	6% (481/7252)

Functions

LOCpro	16% (18624/110074)
V	22% (25252/110074)

Management Controlling Review Development

TestLinux/Session_4	Alarms	v(G)	LOCphy	LOCpro	LOCcom	V	B	Mi
Measured files		72381	465728	5310836	3777711	996065		105
11d.c		23	54	698	473	111	19397.143	102
21142.c		11	70	258	208	37	10923.936	79
21285.c		14	58	517	386	48	15676.426	111
2860_main_dev.c		30	98	1377	801	362	33169.863	110
2860_rtmp_init.c		22	57	922	581	200	28464.264	96
2870_main_dev.c		39	140	1612	1018	372	47315.004	103
2870_rtmp_init.c		46	143	1778	1133	379	55891.031	87
3780i.c		19	22	740	486	81	20047.533	95
3c359.c		49	153	1811	1143	370	70714.203	94
3c501.c		11	71	899	428	370	17770.238	109
3c503.c		21	91	756	542	152	24685.988	100
3c501.c		215	215	1665	1028	439	56411.895	104
3c507.c		22	81	938	586	265	29948.424	101
3c509.c		48	171	1586	1162	251	57740.031	103
3c515.c		44	211	1579	1190	311	65548.945	89
3c523.c		34	117	1307	858	325	49440.996	100
3c527.c		23	87	1659	842	571	39736.063	109
3c574_cs.c		35	104	1223	857	221	48854.074	101
3c589_cs.c		29	104	994	720	156	39841.961	103
3c59x.c		84	433	3243	2467	506	155505.094	88
3w-9xxx.c		76	247	2224	1646	319	93169.773	95
3w-xxxx.c		77	222	2435	1698	464	89916.898	98
53c700.c		53	232	2175	1531	389	86215.078	94
53c700.h		2	23	524	404	93	18507.627	103
68328f.c		12	55	502	344	115	15669.361	99

D:\linux-2.6.29.4\drivers\net\3c503.c

```

/* 3c503.c: A shared-memory NS8390 ethernet driver for linux. */
/*
Written 1992-94 by Donald Becker.

Copyright 1993 United States Government as represented by the
Director, National Security Agency. This software may be used and
distributed according to the terms of the GNU General Public License,
incorporated herein by reference.

```

show details

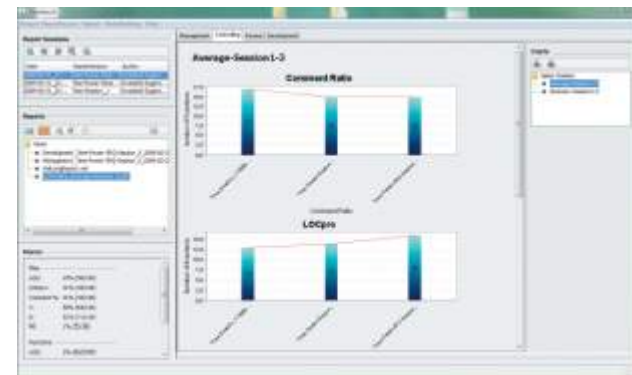
Testwell CMT++ / CMTJava

Controlling View

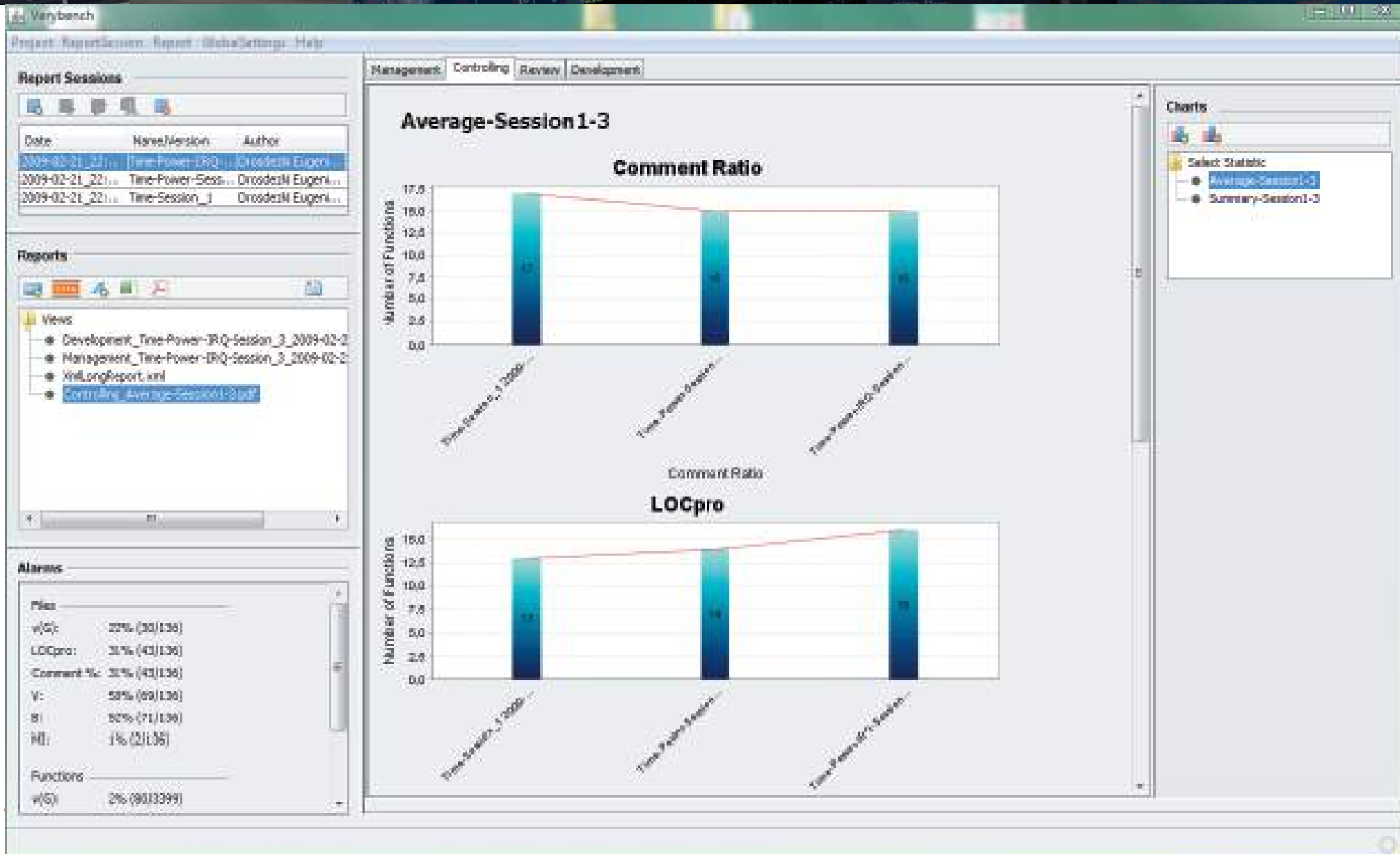
Compares different reporting sessions

-> visualize changes of a complexity's behavior over time

With SVN support enabled it is even possible to follow a behavioral pattern between revisions.



Testwell CMT++ / CMTJava



Testwell CMT++ / CMTJava



Further information:

www.verifysoft.com