Verifysoft TECHNOLOGY

Testwell CTC++

Testwell CTC++ Test Coverage Analyser

What is Testwell CTC++?

Testwell CTC++ is the leading Code Coverage Tool for measuring Code Coverage on host and on all embedded targets (even very small ones). The tool can be used to fulfill the code coverage requirements of safety standards like DO-178C, ISO 26262, EN 50128, and IEC 60880.
Hundreds of companies in more than 30 countries all over the world use Testwell CTC++ in order to insure the quality of their softwares. Testwell CTC++ is the first choice for companies which have to achieve and to proof high code coverage in aerospace, automotive, transportation, healthcare, nuclear power and other industries.

Why Code Coverage?

Code coverage is a measure, which describes the degree to which the source code of a program is tested. This can be considered as an indirect measure of quality of the software.
Code coverage finds areas of a program which have not yet exercised by a set of test cases. This way it supports you by creating additional test cases to increase the code coverage and prevents you from writing redundant test cases. Code coverage is most useful during the module testing phase, though it also has benefit during integration testing and at other times, depending on how and what you are testing.
Code coverage is « highly recommanded » (which means de facto mandatory) for safety critical software. Safety standards like **DO-178C** (Software Considerations in Airborne Systems and Equipment Certification), **IEC/EN 61508** (functional safety of electrical/electronic, programmable electronic safety-related systems), **EN 50128** (Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems), **IEC 60880** (nuclear power), **and ISO 26262** (functional safety of road vehicles) request different levels of code coverage according to the safety level of the application.
But also for the development of « non-safety-critical software, quality matters. Code coverage helps to insure highest code quality.

www.verifysoft.com

Proven success of Testwell CTC++

Testwell CTC++ Test Coverage Analyser is successfully used in the industry since 1989 already. The initial versions have been developed within the NOKIA group at the Tampere (Finland) site of companies Oy Softplan Ab, Nokia Data Systems Oy and ICL Personal Systems Oy. In 1992 the company Testwell has been founded with the mission to further develop, market and support the "Testwell" tools. Testwell´s founder Mr. Olavi Poutanen has been in charge of the development of these tools already at the time he was working with the above mentioned companies.
After ten years of successful sales in Europe, Verifysoft Technology GmbH from Offenburg (Germany) aquires Testwell CTC++ in 2013.
Today hundreds of customers from over 30 countries on all continents use Testwell CTC++ with great success.

Why is Testwell CTC++ that successful?

Testwell CTC++ is the first choice for analysing Code Coverage.
The tool is very easy to use. It has a broad language support (C, C++, Java and C#) and analyses all coverage levels up to Modified Condition/Decision Coverage (MC/DC) and Multicondition Coverage. Testwell CTC++ works with all compilers and cross-compilers. The tool has a very low instrumentation overhead. Thanks to the support of all embedded targets and microcontrollers, Testwell CTC++ is widely used for the development of embedded software.
The tool is integrated in many IDEs and a lot of tool chains and testing environments. The coverage reports are clear and meaningful.
Testwell CTC++ has been compared to US Army Jeep: Simple to use. Works. Can be driven on almost whatever terrain (especially when Hoast-Target add-on of Testwell CTC++ is used). You are looking for a Code Coverage tool which works in any situation? **Testwell CTC++ is your best choice!**
Testwell CTC++ is used for safety critical development. It is compliant with safety standards like DO-178C, ISO 26262, IES 60880, and IEC 61508. For ceritfication purposes, we deliver a Tool Qualification Kit for Testwell CTC++.

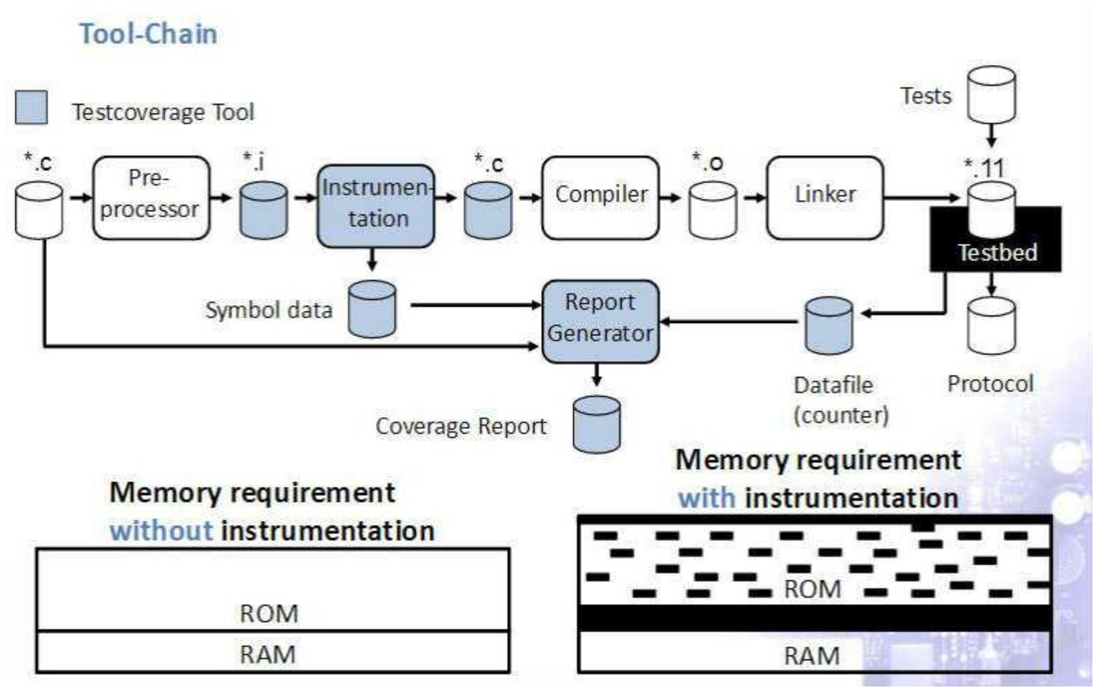
Testwell CTC++ successfully used in Transportation Industries
Compliant with EN 50128
ABB
BOMBARDIER
SIEMENS

How does Testwell CTC++ work?

Testwell CTC++ Test Coverage Analyser is very easy to use. No modifications of the user's code is needed. The code coverage analysis is done by code instrumentation. Execution counters which are automatically added to the source code, count how many times a part of the code has been executed (tested). Instrumentation takes place by just adding ´ctc´ in front of the compilation/link command.

Enforcing the existing makefiles to build instrumented targets instead of the original non-instrumented ones is very straightforward. This does not need any changes to the makefiles itself.
Testwell CTC++ can be used via **command-line** or directly from many IDEs.
When used in command-line mode (by makefiles or other build scripts), the instrumentation is just a front end phase at the compile/link command. **No changes to the source files or build scripts are needed.** Test runs are done with the instrumented program version, in the same way as with the original program.
As Testwell CTC++ has a **very low instrumentation overhead** on the size and execution speed, and special technical concepts (host-target coverage, bit-coverage) for analysing on-target test coverage, it **can be used for all (even smallest) embedded targets** and microcontrollers.
Coverage and other execution profiling reports are obtained easily in straight text, HTML, XML and Excel input form. The **reporting is informative and well-organised**. The reports give both a top-level view, which show the coverage percentages at various summary levels, and a detailed view.

Browsing the coverage results in HTML is very easy. The overall picture is shown in color-coded histograms of coverage percentages. Zooming to the detailed level can be be done with only a few mouse clicks and the untested code locations are clearly shown **mapped to the original source code**.
Testwell CTC++ can be seamlessly used with other vendors´ unit test and system (GUI) test execution frameworks. Our coverage tool works with all unit testing tools.



Support of all Compilers/Cross-Compilers

Each single license of Testwell CTC++ supports **all compilers** and cross-compilers. Even new compilers which you might use in future will be supported by your CTC++-license. No need to spend additional money to get Testwell CTC++ work with new compilers.

We have ready settings for a large number of compilers.
The current list (as of June 2016) is as follows:

Altium Tasking: cc166, ccm16c
Borland/Inprise/Paradigm/Codegear compilers: bcc, bcc32, pcc, pcc32 (Paradigm)
Cosmic compilers: cx6805, cx6808, cx6812, cxs12x, cxxgate, cx6811, cx6816, cx332, cxst10, cxstm8, cxst7
gcc and all gcc based cross-compilers: i586-mingw32msvc-gcc, x86_64-linux-gnu-gcc, m68k-palmos-coff-gcc, tricore-gcc, arm-linux-gnueabi-gcc., arm-none-eabi-gcc, arm-none-linux-gnueabi-gcc, arm-elf-gcc, arm-montavista-linux-gnueabi-gcc, pic30-gcc, pic32-gcc, avr-gcc, xc16-gcc, mlx16-gcc, thumb-epoc-pe-gcc, arm4-epoc-pe-gcc, armv-epoc-pe-gcc, powerpc-wrs-linux-gnu-e500v2-glibc_small-gcc, *-gcc, *-*-gcc, *-*-*-gcc, HPUX CC, HP C++, aCC
IAR compilers and toolchains: iccm16c, icc430, icc8051, iccarm, iccavr, iccavr32, icccf, icchcs12, iccmaxq, iccdspic, iccpic18, icccr16c, icc78k, icc78k0r, iccv850, icch8, iccm32c, iccr32c, iccsam8
Fujitsu/Softune: fcc907s, fcc911s
GHS/GreenHills/Multi: ccv850, cxv850, ccmips, cxmips, ccarm, cxarm, ccppc, cxppc, gcc (GreenHill, not GNU)
Hitachi: shc, shcpp, ch38, ccrx
HI-Tech PICC compilers (Windows and Linux): picc, picc18, picc32, dspicc
VisualDSP++: ccblkfn, cc21k, ccts
Intel compilers (all platforms): icc, ic86, ic96
Java compilers: javac, jikes, ecj, gcj, kaffe
Keil compiler: c51, c166, c251, ca, armcc
Matlab/Simulink: lcc
Metaware: hcarm and others
Microsoft compiler: cl on host both 32 and 64 bit, cl for Smartphones and PocketPC, csc C# compiler, vjc J# compiler
Mitsubishi: nc30, nc308, nc77, nc79
Mono compilers: mcs, gmcs, smcs, dmcs
Motorola: chc12
NEC: ca830, ca850
Pathscale pathcc/pathCC
Renesas: shc, shcpp, ch38, ccrx, nc, nc308, nc77, nc79A, cc32R, CS+/CubeSuite+ cc78k0, cc78k0r, cx, ca850
Sun compilers: WorkShop compilers, javac
Symbian: various compilers
TI Code Composer Studio: cl2000
Trimedia: tmcc
Windriver: ccarm, ccsimpc, g++simpc, g++arm, cchppa, ccsimso, ccsparc, cc68k, cc386, cc960, ccmips, ccppc

Please note that our customers have run Testwell CTC++ also with other compilers. **Adaptations to further compilers are easy and free of charge for our customers**. It can even be done by yourself.

Further compiler settings are added as soon as there is a customer need. Please check http://www.verifysoft.com/en_code_coverage_all_compilers.html for an updated list.

Integration in many IDEs

Testwell CTC++ Test Coverage Analyser is integrated / interfaced in many Tool Chains, Testing Environments, and Software Quality Tools.
Testwell CTC++ Test Coverage Analyser is currently (June 2016) integrated in the following IDEs: Visual Studio, IAR-Workbench, Borland 5.02, BeckIPC, Eclipse, Fujitsu Softune, Renesas. The tool works also with Keil µVision IDE (- c51, C166, c251, ca, -armcc), and ARM DS-5 armcc.
It is possible to make Testwell CTC++ work with other IDEs, even if they are not listed here.
The requirements of such an integration are as follows : Editiable Commandlinepattern, Editiable Makefile-generation, Modifiable Tools (like Code::Blocks-Tools). If one or more of this preconditions are met, you could integrate CTC++ yourself. A video explaining how to integrate Testwell CTC++ in your IDE is available here : https://www.youtube.com/watch?v=Pen_YNk_fQ0&feature=youtu.be
Please contact us if you are interested in an integration of Testwell CTC++ into other IDEs.

Integration in many Tool Chains and Testing Environments

Testwell CTC++ Test Coverage Analyser is currently (June 2016) integrated in with the following Tool Chains, Testing Environments, and Software Quality Tools:

- CATIA Systems – AUTOSAR Builder (Dassault Systemes)
- dSpace SystemDesk
- dSpace TargetLink
- Imagix 4D
- Jenkins
- Lauterbach
- MATLAB Simulink
- PikeTec Time Partition Testing (TPT)
- SonarQube
- QTronic TestWeaver
- QTronic Silver

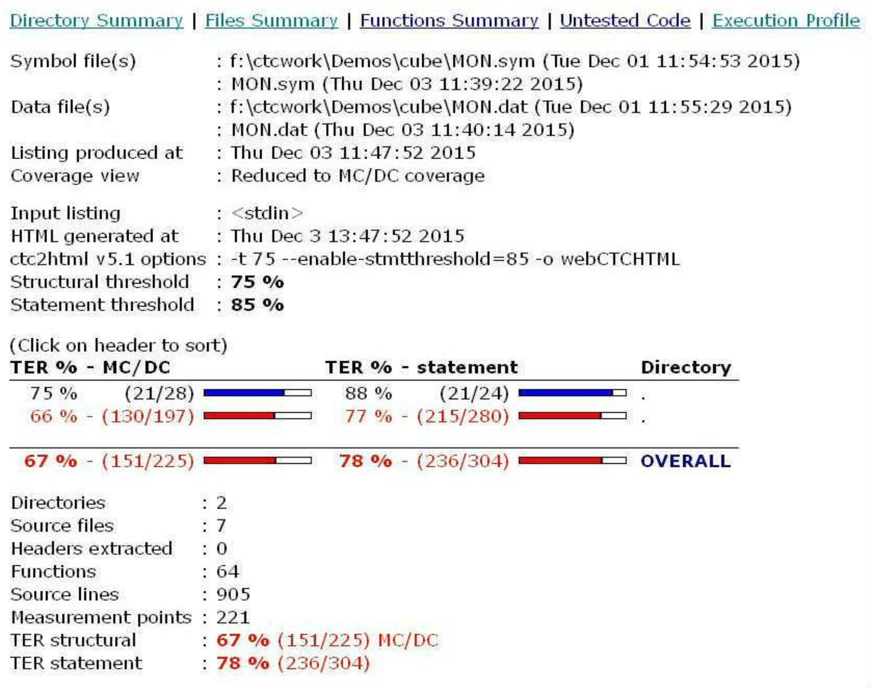
Please contact us if you are interested in an integration of Testwell CTC++ into other tool chains and testing envrionments. Verifysoft Technology GmbH is also interested in partnerships with other tool vendors wishing to get Testwell CTC++ integrated into their tool(s).
Please note that Testwell CTC++ works together with almost all unit test tools.

Clear and Meaningful Reports

Testwell CTC++ analyses Code Coverage for all coverage levels up to MC/DC and Multicondition Coverage and shows the analysis results in clear and meaningful reports. The reports give both a top-level view, which show the coverage percentages at various summary levels, and a detailed view, where the executed/not executed information is mapped to the actual source code locations.

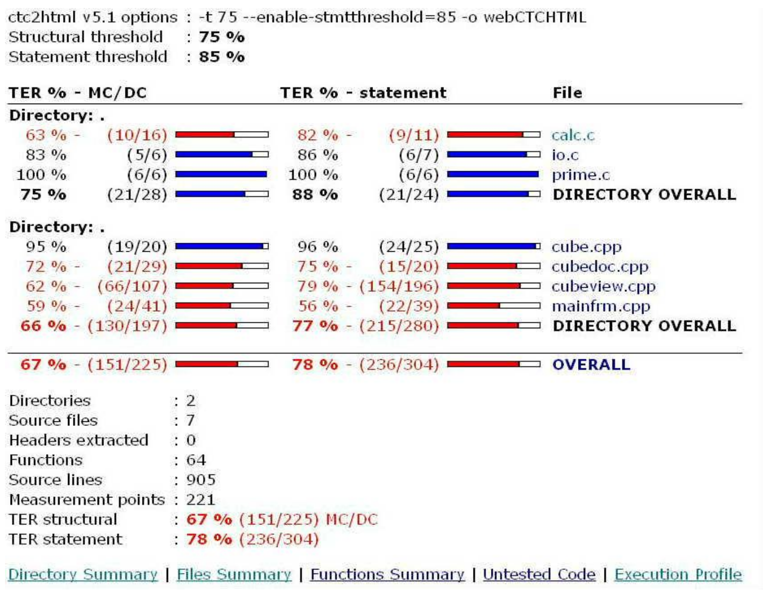
CTC++ Coverage Report (HTML format, hierarchical with 4 levels)

- Directory Summary (General header information)



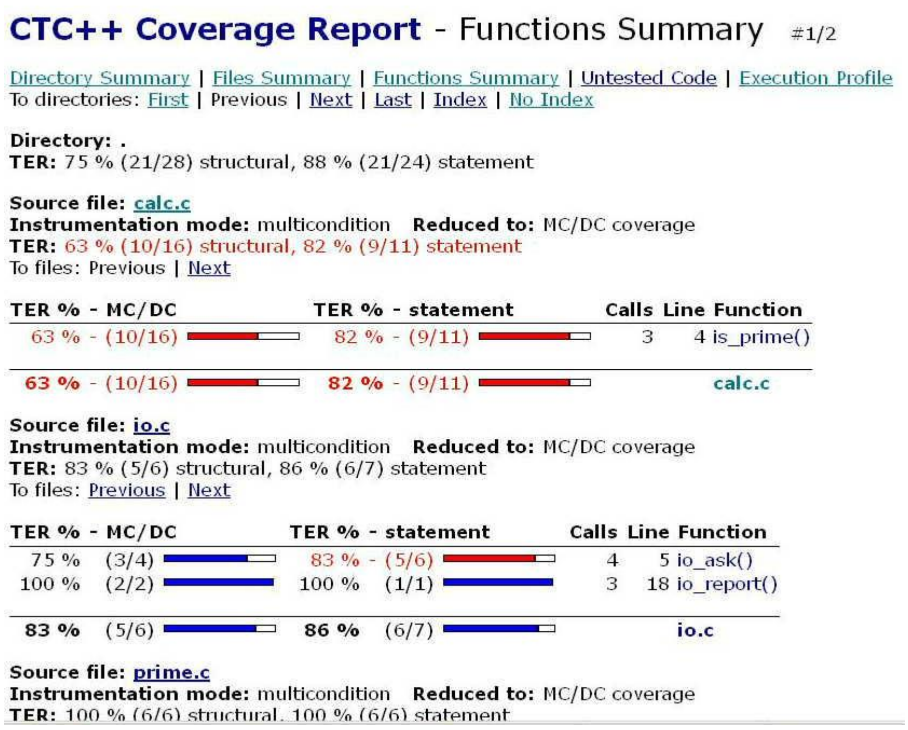
```
Directory Summary | Files Summary | Functions Summary | Untested Code | Execution Profile

Symbol file(s)          : f:\ctcwork\Demos\cube\MON.sym (Tue Dec 01 11:54:53 2015)
                        : MON.sym (Thu Dec 03 11:39:22 2015)
Data file(s)            : f:\ctcwork\Demos\cube\MON.dat (Tue Dec 01 11:55:29 2015)
                        : MON.dat (Thu Dec 03 11:40:14 2015)
Listing produced at     : Thu Dec 03 11:47:52 2015
Coverage view           : Reduced to MC/DC coverage

Input listing           : <stdin>
HTML generated at       : Thu Dec 3 13:47:52 2015
ctc2html v5.1 options   : -t 75 --enable-stmtthreshold=85 -o webCTCHTML
Structural threshold    : 75 %
Statement threshold     : 85 %

(Click on header to sort)
TER % - MC/DC                  TER % - statement              Directory
  75 %        (21/28)            88 %        (21/24)          .
  66 % - (130/197)               77 % - (215/280)             .

 67 % - (151/225)               78 % - (236/304)              OVERALL

Directories            : 2
Source files           : 7
Headers extracted      : 0
Functions              : 64
Source lines           : 905
Measurement points     : 221
TER structural         : 67 % (151/225) MC/DC
TER statement          : 78 % (236/304)
```

- Files Summary (Zoom-in to the files in the directories)



```
ctc2html v5.1 options  : -t 75 --enable-stmtthreshold=85 -o webCTCHTML
Structural threshold   : 75 %
Statement threshold    : 85 %

TER % - MC/DC              TER % - statement            File
Directory: .
  63 % -    (10/16)          82 % -     (9/11)          calc.c
  83 %       (5/6)           86 %       (6/7)           io.c
 100 %       (6/6)          100 %       (6/6)           prime.c
  75 %      (21/28)          88 %      (21/24)          DIRECTORY OVERALL

Directory: .
  95 %      (19/20)          96 %      (24/25)          cube.cpp
  72 % -    (21/29)          75 % -    (15/20)          cubedoc.cpp
  62 % -   (66/107)          79 % - (154/196)           cubeview.cpp
  59 % -    (24/41)          56 % -    (22/39)          mainfrm.cpp
  66 % - (130/197)           77 % - (215/280)           DIRECTORY OVERALL

  67 % - (151/225)           78 % - (236/304)           OVERALL

Directories          : 2
Source files         : 7
Headers extracted    : 0
Functions            : 64
Source lines         : 905
Measurement points   : 221
TER structural       : 67 % (151/225) MC/DC
TER statement        : 78 % (236/304)

Directory Summary | Files Summary | Functions Summary | Untested Code | Execution Profile
```

- Functions Summary
  Zoom-in to the methods and functions in the files



CTC++ Coverage Report - Functions Summary #1/2

Directory Summary | Files Summary | Functions Summary | Untested Code | Execution Profile
To directories: First | Previous | Next | Last | Index | No Index

Directory: .
TER: 75 % (21/28) structural, 88 % (21/24) statement

Source file: calc.c
Instrumentation mode: multicondition **Reduced to:** MC/DC coverage
TER: 63 % (10/16) structural, 82 % (9/11) statement
To files: Previous | Next

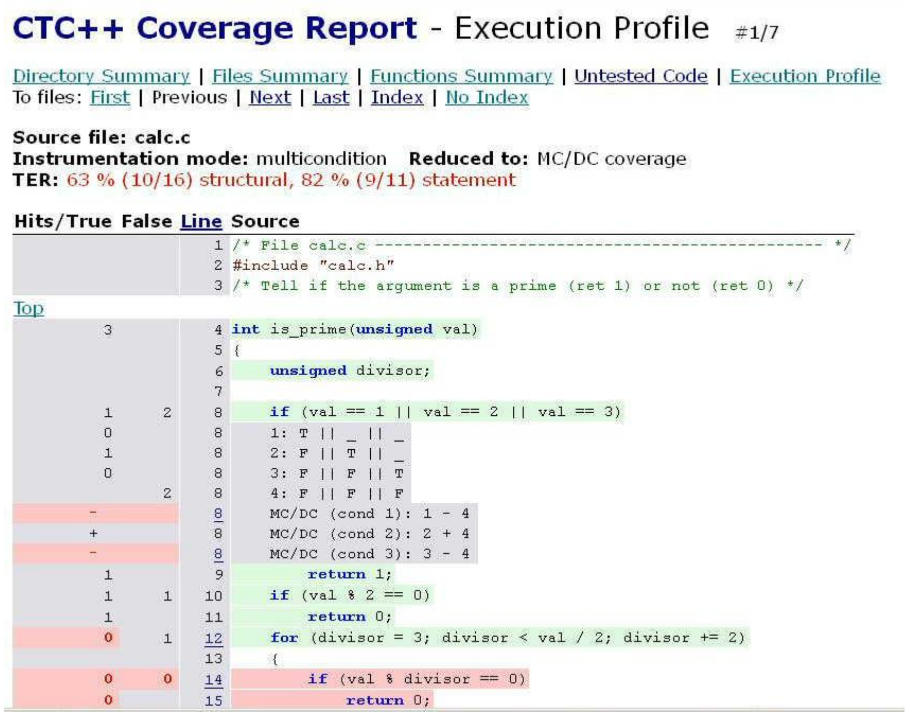
TER % - MC/DC		TER % - statement		Calls	Line	Function
63 % - (10/16)	🟥	82 % - (9/11)	🟥	3	4	is_prime()
63 % - (10/16)	🟥	**82 %** - (9/11)	🟥			calc.c

Source file: io.c
Instrumentation mode: multicondition **Reduced to:** MC/DC coverage
TER: 83 % (5/6) structural, 86 % (6/7) statement
To files: Previous | Next

TER % - MC/DC			TER % - statement			Calls	Line	Function
75 %	(3/4)	🟦	83 % - (5/6)		🟥	4	5	io_ask()
100 %	(2/2)	🟦	100 %	(1/1)	🟦	3	18	io_report()
83 %	(5/6)	🟦	**86 %**	(6/7)	🟦			io.c

Source file: prime.c
Instrumentation mode: multicondition **Reduced to:** MC/DC coverage
TER: 100 % (6/6) structural, 100 % (6/6) statement

- Execution Profile
  Zoom-in to the detailed view
  execution counters are shown with the source code
  not fully executed lines are shown in red



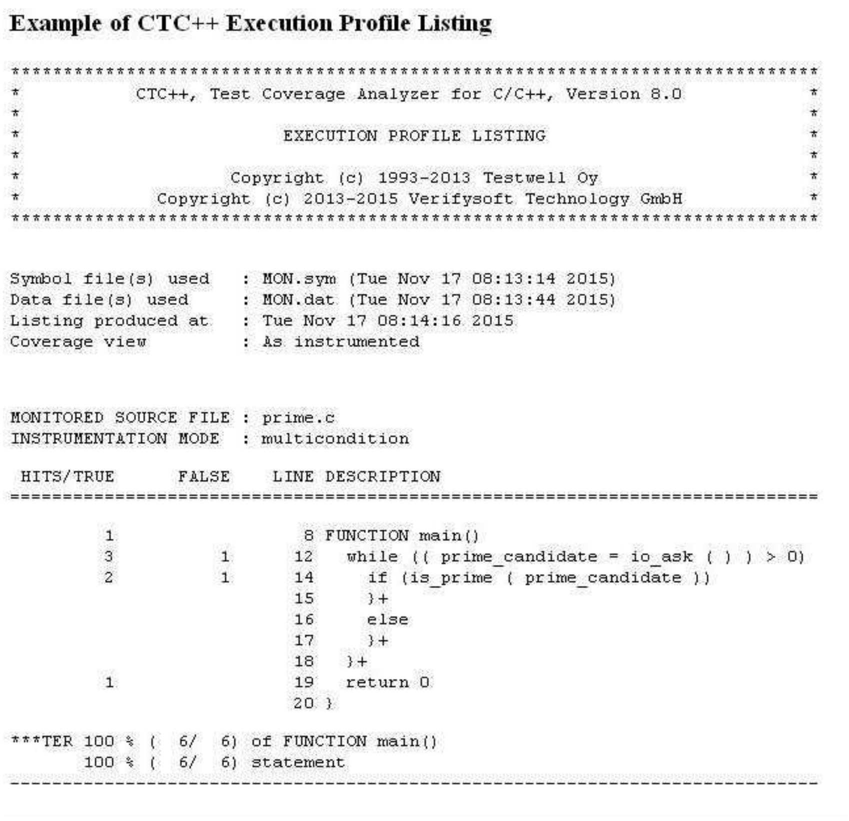
CTC++ Coverage Report - Execution Profile #1/7

Directory Summary | Files Summary | Functions Summary | Untested Code | Execution Profile
To files: First | Previous | Next | Last | Index | No Index

Source file: calc.c
Instrumentation mode: multicondition **Reduced to:** MC/DC coverage
TER: 63 % (10/16) structural, 82 % (9/11) statement

```
Hits/True False Line Source
                  1 /* File calc.c ----------------------------------------------- */
                  2 #include "calc.h"
                  3 /* Tell if the argument is a prime (ret 1) or not (ret 0) */
Top
     3            4 int is_prime(unsigned val)
                  5 {
                  6     unsigned divisor;
                  7
     1      2     8     if (val == 1 || val == 2 || val == 3)
     0            8     1: T || _ || _
     1            8     2: F || T || _
     0            8     3: F || F || T
            2     8     4: F || F || F
    -             8     MC/DC (cond 1): 1 - 4
    +             8     MC/DC (cond 2): 2 + 4
    -             8     MC/DC (cond 3): 3 - 4
     1            9         return 1;
     1      1    10     if (val % 2 == 0)
     1           11         return 0;
     0      1    12     for (divisor = 3; divisor < val / 2; divisor += 2)
                 13     {
     0      0    14         if (val % divisor == 0)
     0           15             return 0;
```

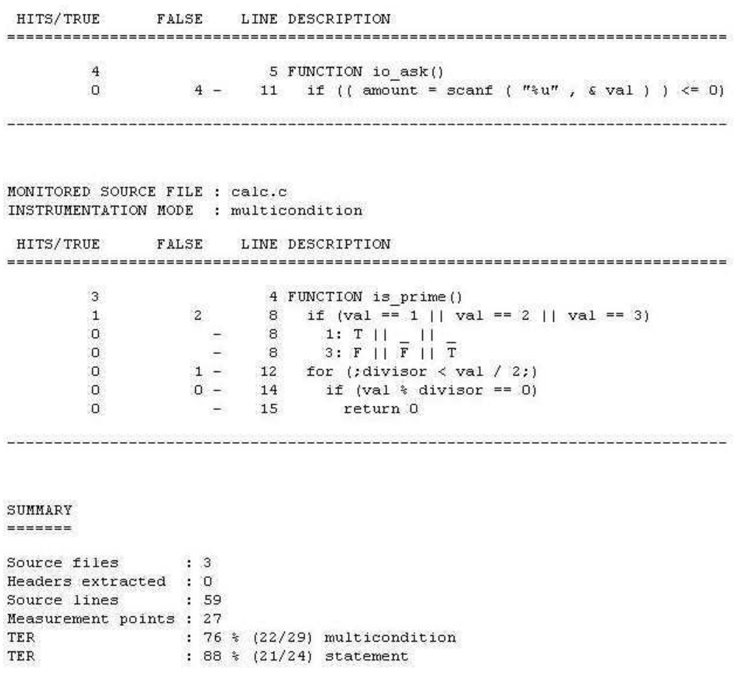
Execution Profile Listing
shows how much time the each code part has been executed
shows the parts which have not yet been executed during testing (textual report)



Example of CTC++ Execution Profile Listing

```
*****************************************************************************
*           CTC++, Test Coverage Analyzer for C/C++, Version 8.0            *
*                                                                           *
*                         EXECUTION PROFILE LISTING                         *
*                                                                           *
*                    Copyright (c) 1993-2013 Testwell Oy                    *
*              Copyright (c) 2013-2015 Verifysoft Technology GmbH           *
*****************************************************************************


Symbol file(s) used   : MON.sym (Tue Nov 17 08:13:14 2015)
Data file(s) used     : MON.dat (Tue Nov 17 08:13:44 2015)
Listing produced at   : Tue Nov 17 08:14:16 2015
Coverage view         : As instrumented


MONITORED SOURCE FILE : prime.c
INSTRUMENTATION MODE  : multicondition

 HITS/TRUE      FALSE    LINE DESCRIPTION
=============================================================================

         1                  8 FUNCTION main()
         3          1      12   while (( prime_candidate = io_ask ( ) ) > 0)
         2          1      14     if (is_prime ( prime_candidate ))
                           15     }+
                           16     else
                           17     }+
                           18   }+
         1                 19   return 0
                           20 }

***TER 100 % (   6/   6) of FUNCTION main()
       100 % (   6/   6) statement
-----------------------------------------------------------------------------
```

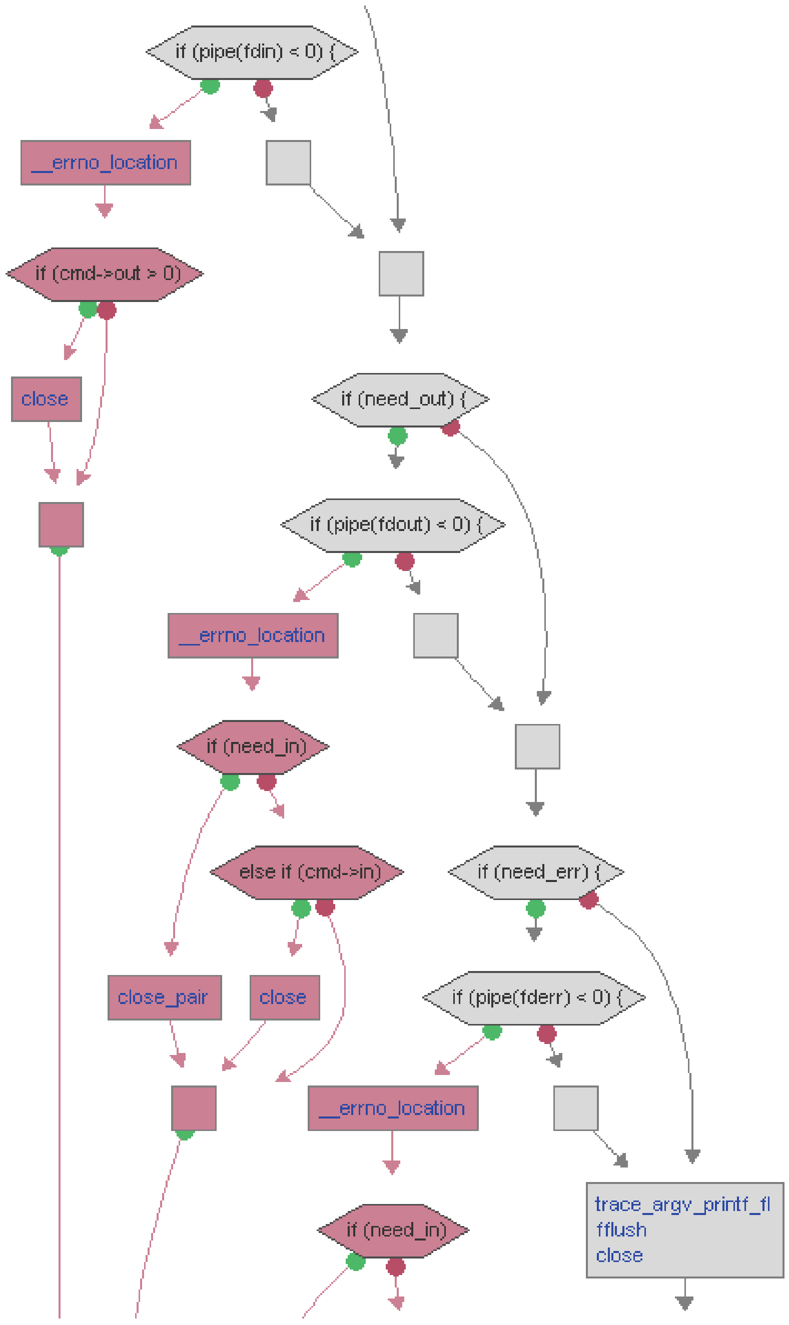
Untested Code Listing
shows the untested code parts (textual report)



```
 HITS/TRUE      FALSE    LINE DESCRIPTION
=============================================================================

         4                  5 FUNCTION io_ask()
         0          4 -    11   if (( amount = scanf ( "%u" , & val ) ) <= 0)

-----------------------------------------------------------------------------


MONITORED SOURCE FILE : calc.c
INSTRUMENTATION MODE  : multicondition

 HITS/TRUE      FALSE    LINE DESCRIPTION
=============================================================================

         3                  4 FUNCTION is_prime()
         1          2       8   if (val == 1 || val == 2 || val == 3)
         0            -     8     1: T || _ || _
         0            -     8     3: F || F || T
         0          1 -    12   for (;divisor < val / 2;)
         0          0 -    14     if (val % divisor == 0)
         0            -    15       return 0

-----------------------------------------------------------------------------


SUMMARY
=======

Source files       : 3
Headers extracted  : 0
Source lines       : 59
Measurement points : 27
TER                : 76 % (22/29) multicondition
TER                : 88 % (21/24) statement
```

Execution Time Listing
shows the cumulative and average execution time of functions
(textual report)

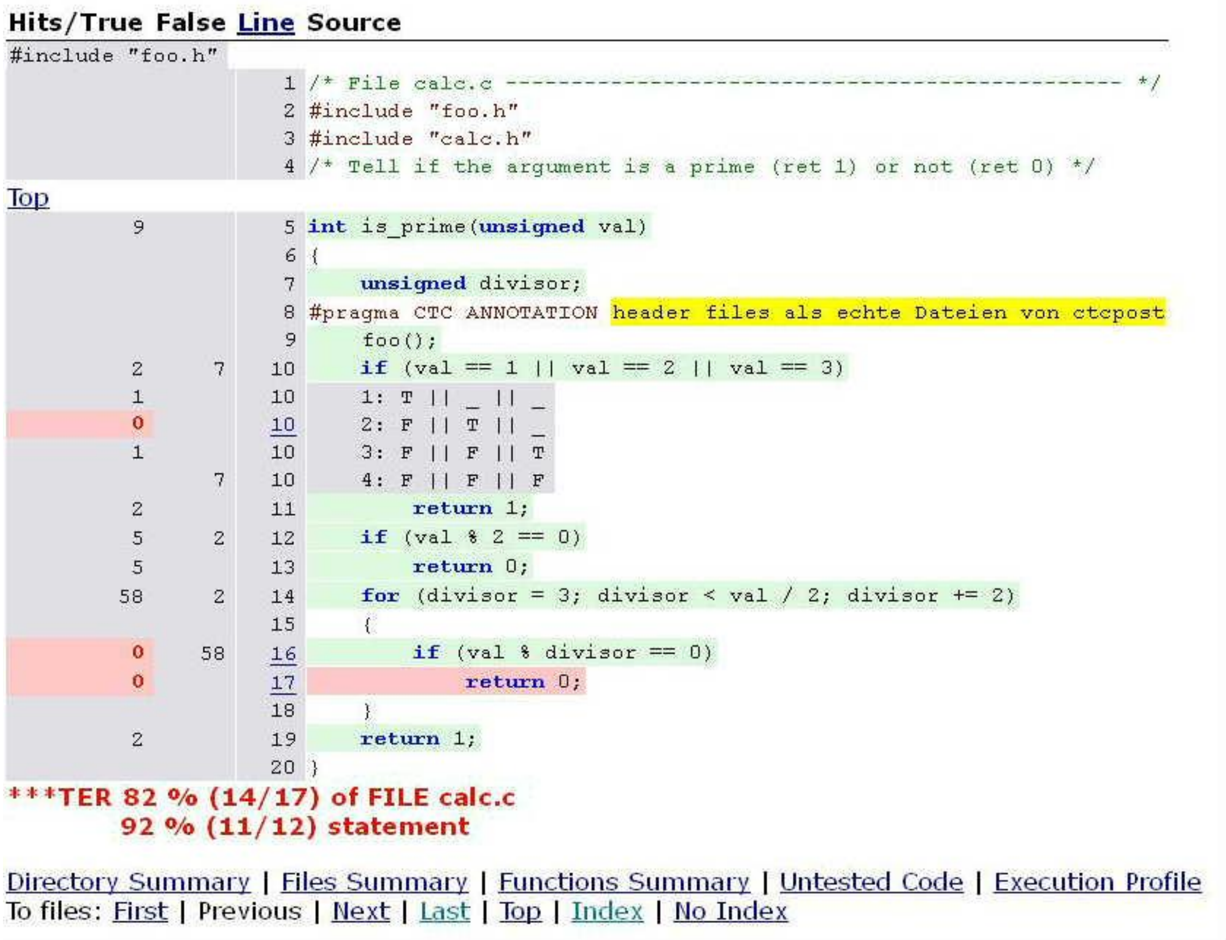
Control Flow Graph (Imagix 4D)
By using Testwell CTC++ together with Imagix 4D the coverage data is shown in a control flow graph.



In this logic layout diamond-shaped symbols represent decisions. Blocks of in-line code are contained in rectangles. Unexercised code is shown in red shapes and paths.

Annotations in Coverage Reports

With Testwell CTC++ you can introduce commentary text to source code, which survive through CTC++ processing all the way to the HTML report. Annotations can be used e.g. to "explain away" why some code portions are not executed in tests (c.f. image below, yellow text).



```
Hits/True False Line Source
#include "foo.h"
                  1 /* File calc.c ----------------------------------------------- */
                  2 #include "foo.h"
                  3 #include "calc.h"
                  4 /* Tell if the argument is a prime (ret 1) or not (ret 0) */
Top
        9         5 int is_prime(unsigned val)
                  6 {
                  7     unsigned divisor;
                  8 #pragma CTC ANNOTATION header files als echte Dateien von ctcpost
                  9     foo();
        2     7  10     if (val == 1 || val == 2 || val == 3)
        1        10     1: T || _ || _
        0        10     2: F || T || _
        1        10     3: F || F || T
              7  10     4: F || F || F
        2        11         return 1;
        5     2  12     if (val % 2 == 0)
        5        13         return 0;
       58     2  14     for (divisor = 3; divisor < val / 2; divisor += 2)
                 15     {
        0    58  16         if (val % divisor == 0)
        0        17             return 0;
                 18     }
        2        19     return 1;
                 20 }
***TER 82 % (14/17) of FILE calc.c
       92 % (11/12) statement

Directory Summary | Files Summary | Functions Summary | Untested Code | Execution Profile
To files: First | Previous | Next | Last | Top | Index | No Index
```

Performs Kernelcoverage

With Testwell CTC++ you can analyse the Test Coverage of a running kernel (Kernelcoverage).

Measuring coverage in kernel-space code is generally a challenge to instrumentation-based coverage tools. In kernel-space code the instrumented probes cannot use any library functions or systems calls while in use-space code they can be used. The CTC++ way of doing the instrumentation and its run-time support layer assumes only basic C with no usage of system services. Thus it can be smoothly executed in kernel space. As a proof of concept for Kernelcoverage we have instrumented a complete Linux kernel and have run a certain test suite on it.

Broad language support

Testwell CTC++ supports **C, C++, Java and C#** programming languages. Initially Testwell CTC++ was developped to support C and C++. Since 2007 we provide add-ons to extend the funciontionalities to Java and C#.

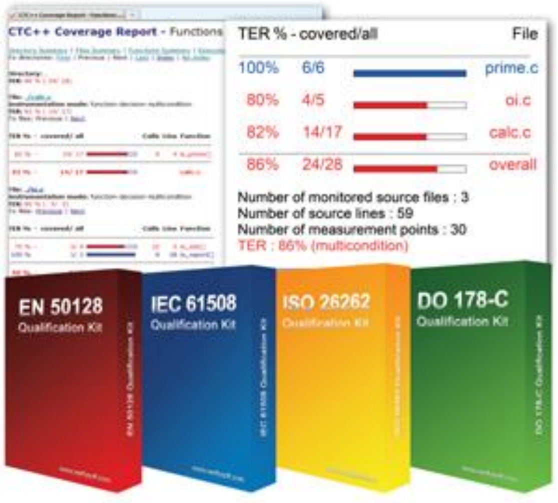
Tool Qualification-Kit Available

The Qualification Kit for Testwell CTC++ provides documentation, test cases, and procedures that let you qualify Testwell CTC++ for projects based on the safety standards **ISO 26262, IEC 61508, EN 50128, IEC 60880 and DO-178C**.
The kit contains tool qualification plans, tool operational requirements, and other materials required for qualifying Testwell CTC++ for usage in safety critical projects. For every used feature of Testwell CTC++ the user is able to execute test cases in his environment that demonstrate the absence of errors.
The kit facilitates certification of embedded systems which use Testwell CTC++ for analysing Test Coverage. The user can modify the artifacts in the tool qualification kit for specific projects.

The qualification kit for Testwell CTC++ is compliant to ISO 26262, IEC 61508, EN-50128, and DO-178C. It consists of:

- A qualification support tool that guides the user through the qualification and generates the following documents:
  - Tool Classification Report
  - Tool Qualification Plan/Report
  - Tool Safety Manual
  - Test Plan
- The test automation unit
- The test suite with test cases.
- The user manual of the qualification kit

The Qualification-Kit for Testwell CTC++ is used successfully by important companies in automotive and aerospace sectors. Most of them rate the kit of "best in class".



TER % - covered/all		File
100%	6/6	prime.c
80%	4/5	oi.c
82%	14/17	calc.c
86%	24/28	overall

Number of monitored source files : 3
Number of source lines : 59
Number of measurement points : 30
TER : 86% (multicondition)

What do our Customers say about Testwell CTC++?

We are proud of hundreds of satisfied customers using Testwell CTC++ to insure highest quality of their applications.
Here some of many positive user testimonials:

"Testwell CTC++ is used in one of our projects on embedded targets. It was easy to integrate on the embedded target. It has a well written manual, and Verifysoft has a good customer support."
Anna Andgart, Software Developer, ABB AB, Control Technologies, Sweden

"We are using Testwell CTC++ very intensively during our development and testing of safety-critical software for nuclear power plants. This tool is not only used on host systems but also on different embedded target platforms. The goal is to easily measure statement/decision or MC/DC coverage to satisfy the test requirements of the nuclear domain software standard IEC60880 and the Common Position document "Licensing of safety critical software for nuclear reactors" which states the requirements from the perspective of European nuclear regulators. Support from VerifySoft is very helpful, fast, direct and straight forward."
Thorsten Oertel, Senior Firmware Engineer, AREVA GmbH, Germany

"Missing coverage is mainly based on missing requirements or incomplete test specifications. So we use CTC++ within development of an advanced driver assistance system to find that missing requirements and to complete our test specifications. This helps us to establish a development process according to ISO 26262. It was easy to integrate CTC++ into our existing development toolchain and it provides immediate results."
Michael Kalusche, Project Manager, Bertrandt Ingenieurbüro GmbH, Germany

"We are using Testwell CTC++ on Instrument Clusters embedded software for Module Testing. The big advantages are high repeatability and fast execution. With this tool we can test the complete Software Module on each small change, and not only the modified part. Giving us much safer results in a shorter time."
Iaran Gadotti, R&D Manager, Continental Brasil Indústria Automotiva Ltda., Brasil

"IAV is one of the largest development service providers in the automotive industry developing software for body electronics. We chose Testwell CTC++ because it supports ISO26262, SPICE and ASIL B and because it can easily be integrated into various and different environments of our customers. Testwell CTC++ provides excellent support for IAV's testing activities at module level measuring the code coverage parallel to the host and target platform.
In addition, we integrated Testwell CTC++ into our Continuous Integration and build process to promptly identify gaps in code coverage and to continuously monitor our quality and code coverage. Testwell CTC++ is now an inherent part of the IAV tool chain."
Marko Meyer, Senior Project Manager, IAV GmbH, Germany

"We use CTC++ extensively to accompany our development and testing of safety-critical software on our embedded targets. CTC++ enables us to track down the coverage of several fragments of our software, so we could adjust our tests precisely to cover all function branches. This serves as well as confirmation for certifications at an inspection authority."
Thomas Bartzick, Department of Software Testing, ISH, Germany

"We are using Testwell CTC++ in our aerospace projects. The goal is "Requirements-based test coverage analysis" to satisfy the DO178B test requirements. The tool supports us in our C source code analysis for example to locate dead code. We are able to use CTC++ without any problems."
Michael Görsdorf, Development / R&D Software, Kappa optronics GmbH, Germany

"We are using Testwell CTC++ in our embedded project. It has helped us to discover any uncovered code and control-path, due to incomplete unit test specifications. The integration is simple and customer support is excellent."
Srinivasulu, Project Manager, Knorr-Bremse Technical Center, India

"Testwell CTC++ is an IEC61508 T3 qualified tool which we use for testing embedded applications. We have been using it both for on-target, and on Windows code coverage analysis. The tool is easy to integrate on embedded applications, and the analysis tools generate pretty good reports."
S. Sánchez-Manjavacas, Senior Firmware/FPGA Architect, Kongsberg Maritime, Norway

"We use Testwell CTC++ for measuring the coverage of the unit tests and system tests of our (embedded) targets. The reports are clean, simple and contain what we need. Support from VerifySoft is decent and fast. The CTC tooling can easily be added to simple builds."
Kees Valkhof, Tester, Lely, Netherlands

"We use Testwell CTC++ to detect the code coverage of the embedded software of our medical products on unit-testing level. Testwell CTC++ could be integrated very well in our build-environment and delivers since then fast and reliably the desired information. The support with possible questions was fast and always able to be of assistance. We could recommend Testwell CTC++ without any restrictions."
Heiko Schmidt, Software Team Manager, MAQUET Cardiopulmonary AG, Germany

"REC Global is embedded software development partner to several major automotive suppliers. As Testwell CTC++ is recognized as a tool of choice for software quality assurance by our customers we followed their lead and implemented it also on our projects. We use it for testing of embedded applications. CTC++ reports serve as an objective measure of test quality and help us improve our development process."
Borivoje Dermanovic, Project Manager, REC Global, Croatia

"We use Testwell CTC++ for analysing the test coverage for our unit- and system tests of our safety product which is developped according to IEC61508. The reports created by Testwell CTC++ are clear, simple, and include all necessary information. It was easy to integrate Testwell CTC++ into our development tool chain. Verifysoft provides an excellent customer support."
Thomas Schneider, Senior Software Engineer, Schneider Electric, Germany

"We are implementing test automation on project with fifteen-year-old legacy code which must meet US FDA software standards. Evaluation of test coverage based on functional specification ("requirements-based testing") is inadequate in this situation. Testwell CTC++ allows us to run many test suites and identify those logic paths which are not covered. Testwell CTC++ is an excellent product."
Robert Evans, Software Development Engineer, Siemens Medical Diagnostics, USA

"Volvocars Powertrain uses CTC++ because it supports ISO26262 and SPICE and because it works well with measuring code coverage in our module test platform."
Johannes Foufas, Developer, Volvocars, Sweden

Summary

Hundreds of customers rely on Testwell CTC++ Test Coverage Analyser because of it’s capability to support all embedded targets, all compilers and all coverage levels.

More Information

More information including the latest news about Testwell CTC++ is available from our homepage http://www.verifysoft.com/de_ctcpp.html

For registered webinars and videos about Testwell CTC++ and related safety standards please have a look to http://www.verifysoft.com/en_ctcpp_online_presentations.htm
Currently (June 2016) the following videos are available:

- General Presentations:
  - Testwell CTC++ General Presentation
  - Code Coverage for Embedded Targets
  - Testwell CTC++ Short Introduction (Prime Example)
  - Safety Standards and related Code Coverage Levels
  - ISO 26262 and Code Coverage
- Usage with special IDEs / environments:
  - Usage of Testwell CTC++ with Microsoft Visual Studio 2008 IDE
  - Usage of Testwell CTC++ with IAR Embedded Workbench IDE
  - Usage of Testwell CTC++ for Embedded Targets (demo is based on an Atmel ATmega 328p µController)
  - How to integrate Testwell CTC++ in your IDE?
  - Integration of Testwell CTC++ in Eclipse
  - Testwell CTC++ and MATLAB/Simulink interface example
  - Testwell CTC++ Code Coverage Lauterbach Trace32
  - Usage of Testwell CTC++ with Renesas CS+ IDE
  - Usage of Testwell CTC++ with Microship MPLAB IDE
- Videos for CTC++ Users:
  - Code Coverate on Embedded Targets with HoTa for Testwell CTC++ Users

Any further questions? Please contact your Verifysoft team from

Verifysoft Technology GmbH
In der Spöck 10-12
77656 Offenburg
Germany
qualify @ verifysoft.com
Phone: +49 781 127 8118-0