innovators in agriculture

Verifysoft TECHNOLOGY

Testwell CTC++ User Testimonial

Lely Industries: Our reputation is built on products which work properly

Excellent product quality including high quality software is essential for Lely. Lely's configuration manager Kees Valkhof has given us an insight into the software testing demands of this supplier for agricultural solutions and why they are using Testwell CTC++ for measuring code coverage.

Kees Valkhof, Configuration manager at Lely Industries, Maassluis, The Netherlands

Lely supplies the agricultural sector with a complete portfolio of products including harvesting solutions, barn cleaners, automated feeding systems and milking robots. These products help farmers to maintain profitable and enjoyable farms.

Lely was founded in Maassluis (the Netherlands) in 1948. More than 2,000 employees work day in day to make farmer's lives easier. We have currently 6 production facilities and 7 R&D departments, which play a key role in our capacity for innovation. Our company has 2,550 active patents protecting our innovations. Farmers in over sixty countries work daily with Lely's solutions for use in the barn or on the field.

Milking robots automate milking so that the farmer doesn't need to pick up all the cows twice or three times a day to milk them. Cows don't sleep as much as humans, so farms need to run when humans sleep. This is one reason why milking machines are driven by software: they also have to work during the night.

Software standards are not yet defined in our kind of industry. Although our software doesn't have safety features and there is no risk that people or animals are injured because of a software failure, **we have set our own standards which are very high.** We have to deliver quality software because we want to deliver quality products to our farmers. Our reputation is built of products that work properly.

Lely's milking system collects data per cow on milk production and cow health. This alerts the faremer on time to any changes, allowing him to devote hisr attention to the cows that need it most.

The farmers can go to sleep and know that the next day their cows are milked and fed.

Software in our products measures the milk quality and cleans the milk lines in the milking machine. If this is done in a wrong way, this will result in a severe issue: the cleaning agent or dirty milk will get into the milk tank. Fortunately the factory will check your milk after your delivery, but in case of a mistake you have a complete truck with bad milk and that is costly.

To avoid such losses, the quality of the software in our products is very important.

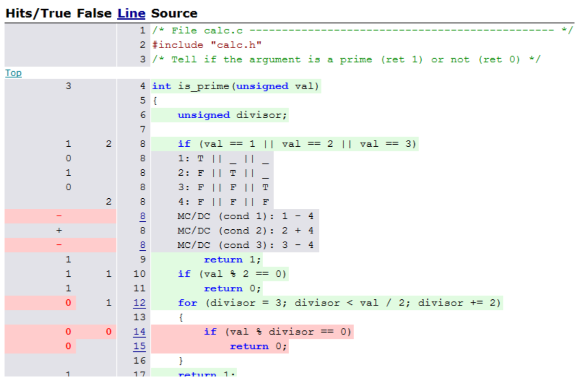
As the systems become more and more complex, there is more and more software. Today we have a lot of software which works together.

With increasing complexity, testing becomes harder and is more work. We have to test your software, automate testing and we have to know what we are testing.

Executing some tests with software is easy. Testing some requirements of software is doable. Verifying all software requirements is a lot of work. **Guaranteeing proper software behavior is a challenge.**

Verifysoft TECHNOLOGY

Even if you can demonstrate that your software performs its intended function, the software can still fail and cause mayhem. Requirements verification does not execute all your code and still leaves sets of states and inputs that might lead to undesired behavior. The trick in finding the buggy untested code is to find these still untested sets of states and inputs. If we know the states and inputs then we can write tests for it and get the uncovered code executed. This is why we are interested in the conditions we need to set during the unit tests. **If we know the conditions, we can cover it.**



```
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;
                  16     }
      1           17     return 1;
```

Testwell CTC++ shows which conditions are missing to cover the source code at 100%.
All coverage levels up to Multicondition Coverage including MC/DC are shown.

At Lely we want to be able to measure the coverage of our unit tests as well as of the tests performed on the target. We have some dedicated embedded boards that require an embedded cross-compiler. As we have more than one of these compilers. we need a generic tooling, which is not bound to a specific tool set and or IDE.

Testwell CTC++

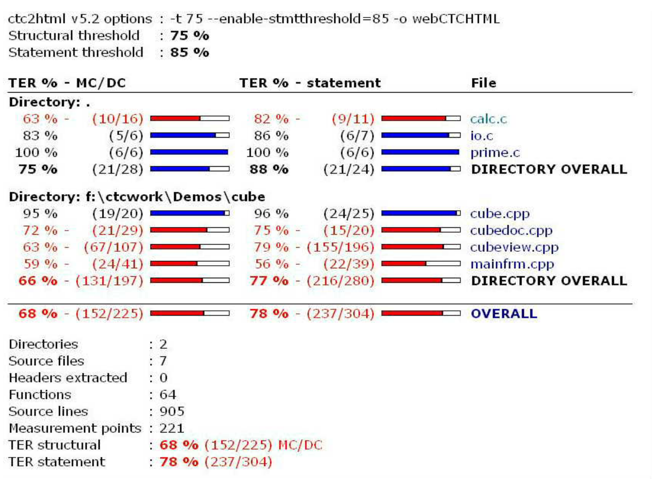
This is the reason why we have chosen in the year 2013 Testwell CTC++ from Verifysoft. This tool meets all of the above requirements.

Testwell CTC++ is completely independent of compilers we are using. During the code instrumentation, Testwell CTC++ adds counters to our source code. This is done before compilation (after pre-compilation) so we can use all our compilers which still work on normal C code.

When you are doing unit tests and you want to explain to your management why you are doing it, you really want to visualize what you actually did. With Testwell CTC++ we can visualize not only what we have tested but also which conditions we have tested. And those conditions we can much easier map to requirements in each line of code.

We use multi-condition coverage and Testwell CTC++ shows us which conditions were tested an which were not. The reports are clean, simple and contain what we need. Nothing more, nothing less.

Verifysoft
TECHNOLOGY



```
ctc2html v5.2 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: f:\ctcwork\Demos\cube
  95 %    (19/20)                 96 %    (24/25)              cube.cpp
  72 % -  (21/29)                 75 % -  (15/20)              cubedoc.cpp
  63 % - (67/107)                 79 % - (155/196)             cubeview.cpp
  59 % -  (24/41)                 56 % -  (22/39)              mainfrm.cpp
  66 % - (131/197)                77 % - (216/280)             DIRECTORY OVERALL

  68 % - (152/225)                78 % - (237/304)             OVERALL

Directories        : 2
Source files       : 7
Headers extracted  : 0
Functions          : 64
Source lines       : 905
Measurement points : 221
TER structural     : 68 % (152/225) MC/DC
TER statement      : 78 % (237/304)
```

The HTML output of Testwell CTC++ gives an overview about
The achieved coverage for each file and each function

The output after testing is also independent of the IDE we are using. This is good because we use multiple ones. The results of the coverage analysis are presented as nice HTML pages which show overall statistics for the whole project, as well as coverage metrics for each file and for each function. We can browse our code, which is annotated with the level of coverage chosen.

We have a complex build that has to compile from multiple compilers we use. So it was a bit work to set Testwell CTC++ up, but the tool is flexible enough to support our complex environment. **We have had a look to other products and they are much harder to integrate** if you are using multiple compilers on your build system.
For Lely, **Testwell CTC++ is a nice tool because it enables us to test what we need to test.**

Kees Valkhof
Lely Industries.
Cornelis van der Lelylaan 1
3147 PB Maassluis
The Netherlands
www.lely.com

Watch the video with
Kees Valkhof on Youtube:

Testwell CTC++ is a tool and a trademark of Verifysoft Technology GmbH
For further questions please visit www.verifysoft.com and contact us at +49 781 127 8118-0