Certifying your Code

Paul E. Bennett [HIDECS Consultancy] http://www.hidecs.co.uk/

Thorough testing and verification of code is often neglected. Paul Bennett introduces us to concepts and procedures that we can all use and concludes that Forth can be used in most safety critical situations. It has been a long held belief, within the software industry generally, that the only programmes that are capable of being 100% tested are "Toy" programmes. I am presuming that this is meant to indicated programmes of under a couple of hundred LOC. Programmes of such a small size would be quite within the realms of any programmer (in whatever language) to fully test in reasonable time. However, let us look at what Forth gives us that is different to many programming environments.

High Integrity Systems, we are assured, have been thoroughly tested. This leaves us all with the unanswered question of "How Thorough is thorough?" Considering that much of the High Integrity software that keep our planes in the sky, run our traffic lights and keep us alive on the operating theatre table are usually many thousands of lines of code can we trust that the testing has been done well enough to give us confidence. What the hardware sector has that the software sector would probably like to have is the ability to attach a "Certificate of Conformity" that can present tangible proof that the system has been developed and tested in accordance with some auditable standard method.

The software tools industry always seem to be promising that the new whizzo software tool is the "Silver Bullet" we have been seeking. Development suites like Rhapsody (from Ilogix) or SPARK-Ada have been touted as the most logical choice for Safety Critical Systems Development because they are proven correct mathematically. Not only are these tools very expensive they also seem to give an unwelcome boost the resources requirements on the embedded system.

In a recent thread on comp.lang.forth Pete < forthsafe@yahoo.com > asked whether or not Forth could be used in Safety Critical Systems and pass the stringent requirements of FDA or DO178B. As one who has been involved in the development and certification of systems that have had to pass such scrutiny I am happy to say that Forth, produced in the right way, can indeed be used for such projects.

In Forth, each word can be considered as a complete programme in its own right. It may be just a subroutine of a much larger programme but in itself it is quite complete and, at the interpreted level, is interactively available for quite thorough testing. Any Forth word can be considered as a fully single minded simple programme in itself. The ultimate "Toy" programme.

Because Forth words, written properly (preferably with reference to a coding standard), are usually quite small and simple we are merely coupling a number of individual, complete, simple programmes.

Each word constructed using other previously tested words would inherit the properties of testing already carried out. The words in a specific lexicological grouping can be seen as existing between programming surfaces that provide the API between functional entities. In this way, the most complex Forth programmes can carry through the testing at a level thatmore or less trivially accomplished.

The idea of programming surfaces establishes a useful reference for situations where application code is developed away from the real target. So long as the certification can fully prove equivalence there should remain no issues (bar hardware timing) that would invalidate the certification of the software.

It was realising this simple facet of Forth that gave me confidence enough to develop the review and testing process to a method for fully certifying Forth code. The method also works with assembly, so long as the assembly subroutines are small, single function, highly coherent and utilise minimal coupling of the systems the code is being developed for.

It should be obvious that the development process by which High Integrity Software is developed for these Safety Critical Systems should be rigorous, and rigorously applied.

Development Management processes that fall below CMMR level 3 are less likely to be trusted well enough for software certification to be believable. Ideally, the organisation should be at level 4 or 5. Therefore, establishing a decent development process is a necessary pre-requisite to being able to produce certifiable software. It is an absolute necessity that the exact source code that is certified is always uniquely identifiable (name, version, production date, test dates etc). A version control and configuration management system, therefore, becomes essential.

Within this article I have included an arbitrary bit of code which should illustrate how the certification process not only improves the presentation of the code but also ensures its logically correct implementation. The form format is made part of the source code file along with the words glossary description and any additional notes. The example code is simple enough to follow without the addition of the glossary comments. However, the incorporation of the glossary text within the source files aids fuller understanding without having to reference other sources.

```
\ ***********************************************************
: .TOS ( S: n1 -- n1 )
( G: Non-destructively print the top of parameter stack item )
( n1 to the current terminal. )
DUP . ;
\ ***********************************************************
\ *   Inspection    *    Function Test    *    Limits Test    *
\ *                 *                     *                   *
\ *                 *                     *                   *
\ ***********************************************************

\ ***********************************************************
: FIB-ALG ( S: n1\n2 -- n2\n3 )
( G: Given two numbers, n1 and n2, of a Fibonacci series )
( calculate the next number, n3, of the series. )
SWAP OVER + ;
\ ***********************************************************
\ *   Inspection    *    Function Test    *    Limits Test    *
\ *                 *                     *                   *
\ *                 *                     *                   *
\ ***********************************************************

\ ***********************************************************
: FIBONACCI ( S: n1\n2\limit -- )
( G: Given the two numbers, n1 and n2, of a Fibonacci series )
( calculate and print the succeeding numbers in the series )
( up to limit to the current terminal. )
>R
BEGIN DUP R@ > NOT
WHILE .TOS FIB-ALG
REPEAT 2DROP R> DROP
\ ***********************************************************
\ *   Inspection    *    Function Test    *    Limits Test    *
\ *                 *                     *                   *
\ *                 *                     *                   *
\ ***********************************************************
```

The three boxes underneath the word definition are intended as an appropriate space in which to collect signatures from the code inspection and test personnel. The three boxes deal with a specific aspect of the inspection and tests to be applied. You can choose whether or not the three boxes are a permanent feature within the source file or generated during documentation print-out. You may even discover a way, within your own configuration management system, to incorporate electronic signatures within these boxes.

Inspection

This is the traditional static inspection run along Fagan Inspection lines. The code inspector should ensure that the code appears to fully implement the requirements stated within the glossary text that is part of the source. This requires that the inspector will need to ensure that the called words used within this definition are also previously certified. Previous reviews of the glossary text should have, of course, ensured that the intent for the word under examination as defined in the glossary text is right for the application. This is one of the benefits of writing the glossary text first.

Function Test

The function test is the first time that the code is formally run. The code might have been exercised by the programmer immediately after coding but this formal run records the performance of the word in comparison to the intent expressed in the glossary text. Several occurrences of the function test would be usual, especially with different representative values, to provide a level of confidence that the function always does as is expected. This test only covers the normal operating range.

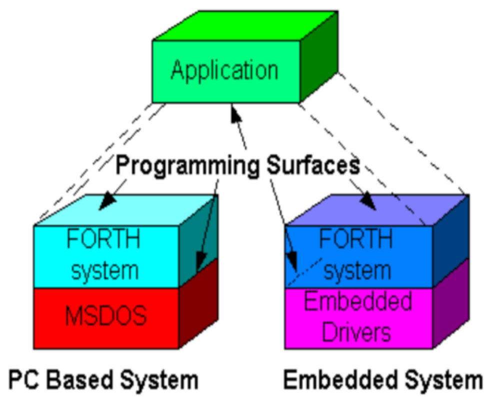
Limits Test

The Limits test is an opportunity to test the code to beyond the expected normal range of operation. If the code is able to deal with wildly out of bounds input stimuli, high rate call demand (if the code is an interrupt) and still behave in a rational manner then this test can be deemed passed. In order to perform this test on your code often requires a very inventive frame of mind to ensure that the operation is, as far as is practicable, out of the normal operational range.

With the code having satisfied the test and inspection personnel, their signatures can be attached to the source print-out. This signed listing then needs storing in a long term archive until the product in which the software exists is no longer in service and an additional five years following its decommissioning.

Programming Surfaces

I have used the term programming surfaces in this article and perhaps now is a good time to explain what they are.



The best description of programming surfaces is perhaps as an interface between underlying machine level code and the application language layer or between the application language layer and the application itself. This idea of surfaces is very useful for demarcation of the various elements of a system which, if the surfaces can be proven equivalent, allow easy movement of segments of code from one platform to another. This is most useful when the underlying processor platform goes out of production and you need to replace it with another similar item of hardware. So long as the machine-level to application language level interface surfaces are proven equivalent there should be very little problem in making the substitution. This also works on a finer grained level throughout Forth.

Programming surfaces are part of a component oriented approach to system development. Components are a firmer, harder edged form of objects that can be comprehended, tested and proven of sufficiently good quality.

Summary

This technique to certification is very simple to apply and, coupled with a suitably robust version and change management process that tracks modifications of the source code, can lead to the production of fully certifiable code that satisfies the most stringent requirements (DO178B, FDA, CE marking etc.).

Just creating source code that incorporates the inspection boxes is not enough. The code has got to qualify by proper inspection and testing methods being applied. Coverage of testing can easily approach 100% of all statements.

Application of a certification process against coding standards and language specification standards will improve the quality factors for the software. This improvement is mainly won through an orderly increase in the level of inspection that the code undergoes. It is also due to an improvement in monitored testing to beyond mere functional parameters.

With this simple certification process in mind I was able to confirm that Forth could indeed be used in the most safety critical projects where a software based solution was a viable choice.

Paul E. Bennett

Paul E. Bennett is an Independent Consultant currently working in the Nuclear Power Industry. He has had experience in many Safety Critical sectors including Petrochemical, Marine, Railway, Medical and Transportation Industries. He has been an advocate of the use Forth in High Integrity Distributed Embedded Control Systems. He has written articles on many topics in Forthwrite including real-world interfacing (see http://www.amleth.demon.co.uk).