# High Integrity Systems CODE

*by Paul E. Bennett IEng MIET, HIDECS Consultancy*

**Abstract:-**

In Phil Koopman`s paper “The Grand Challenge of Embedded System Dependability” he sets out four challenges.

“Four significant challenges in embedded system dependability are:

- embedded-specific security approaches,
- unifying security with safety,
- dealing with composable emergent properties,
- and enabling domain experts to use advanced dependability techniques.”

**[Koopman1]**

This paper will describe the benefits of developing software as components of a system, following on from their interface specification, review/examination, functional testing and limitations testing. It will continue to explain why Forth is one such suitable Component Oriented Development Environment. We will cover the development process (including organisational requirements) and the production of well specified and correctly behaving (according to its specification) software.

## Mechanics

Early Engineering was entirely mechanical in nature. The invention of the wheel and axle, the ramp, the Archimedes Screw, Nut and bolt etc. The early days saw some failures but the engineers of the time learnt how to make the components better and stronger within the constraints they were forced to deal with. They also gained the wisdom to know when to state that the constraints were too restrictive and not workable. For installations where safety was a major concern, the notion of using certified components became widespread. Such components had to be certified as meeting specifications and being compliant with standards in materials inspection and testing. Our mechanical engineering counterarts have had a very extensive head-start. An example of an early standard was the one for screw threads.

## Electrical & Electronics

Slightly newer than the mechanical engineering industry sector, the electrical and electronics industry has managed to assist in the simplification of many machines and systems. Integrating with the mechanics, the electrical and electronic systems also became the subject of closer examination and thus eventual Certification of Conformity (CofC) also became compulsory for Safety Related and High Reliability Systems. Then machines and electronics began to become programmable, leading to the creation of seemingly quite clever machines. Thus was born software. However, some early programmable machines led to the occurrence of disasters.

## Software

When software came along, the early implementers were highly skilled mathematicians. However, the number of people creating software grew rapidly and some of the new software writers were neither mathematicians nor engineers. This led to some sloppy code and systems that would fail regularly. Only recently have some software engineers become interested in creating software that truly is delivered by robust engineering methods. Component Oriented Software is one such engineering methodology.

However, systems are not just software. Everything that makes up that system, including the statement of requirements, the design documents, the operating manuals as well as all the nuts, bolts and logic components are a part of the overall system. Rather than different methods for developing the hardware and software a unified development approach that all involved in the process understand fully becomes the basis for managing a “Component Oriented System Devlopment”.

## Development Process

If you want your systems to have real integrity it is necessary to be very adept in the early development of accurate requirements, that are testable for compliance with the clients real needs, before you launch into writing the design specification. Also, remember that prototyping is only meant to be a tool for exploring the detailed requirements space in order to hone the thinking at that level. The two best models of development still tend to be either the "V" model or Barry Boehm's Spiral Model (applied properly and with the early spins being directed solely at the elicitation of the full and correct requirements). The V model has to be applied as a two-prong approach to initially developing requirements and the System Acceptance Test Cases that prove the requirements have been fully met. Thus you will be developing the final acceptance test clauses in parallel with determining what your system needs to do. It is therefore imperative to finding the right descriptive language in which to express the requirements and allow the test cases to be written (even to the extent of handling the un-expected conditions). Tests have to not only cover the acceptance of the component unit but build to the acceptance of the overall system.

Naturally, to have a robust development process, properly followed such that it meets at least the SEI CMM level 3 **[SEI]**, is paramount to providing systems with certification that will be believed. The organisation needs to properly support the development process through assignment of appropriate roles and responsibilities to suitably qualified and experienced personnel. This means that all the management chain has bought in to such development processes and its proper operation. There needs to be enough discipline in the process to succeed and the application of the process needs to provide sufficient data of its correct application. Thus robust configuration management will be required to form the basis of this development process **[Kelly]**.

Finally, for this section, the documentation you produce has to be fit for the purpose of describing the components and the system to which they will belong. There is a world of difference between good and bad documentation **[Montforton].** You don't want too much or too little documentation, but sufficient **[Koopman2]**.

Wernher von Braun once said "Research is what I am doing when I do not know what I am doing". So, to say you are involved in R&D must mean that you are on a voyage to discover what your client truly needs in a way that he can sign his full agreement with your proposals. You might spend more than 60% of project time getting to the stage that the requirements become complete and testable, but, in the long run it is often much faster and yields better quality of product than to leave testing as an afterthought. However, once you have discovered all the aspects of the system and can complete the full requirements specification, including the safety, security, environmental and aesthetic aspects, then the easier it becomes to establish the best components to utilise within the system.

## C.O.D.E

As stated above, the hardware world has managed to accomplish designs with some of the highest integrity, engendering a high level of trust in such systems. We need such in the software world and some attempts were made at the network level with schemes like .NET and CORBA. These are quite large and complex components in general that have to be built to be host system agnostic. However, in control systems, such a networking level is too high a consideration in resource constrained small controllers prevalent in industrial controls (eg. field sensors and devices).

A Component Oriented Development Environment allows developers to independently create and test individual components. In software the author proposes Forth is such an ideal software Component Oriented Development Environment (C.O.D.E) that it becomes easy to develop a library of software components that can be created, tested and fully certified to similar standards as individual components in the hardware world. A Certificate of Conformity for each and every Forth word that has almost mathematical certainty about its behaviour.

**Why Forth?**

This question has been asked very often and a number of very varied responses have been offered in answer. Why certification works with Forth, though, is down to the underlying Virtual Machine Model (VMM) that is the central embodiment of Forth. Despite, over the years, many Forth language standards being created, the basic underlying VMM is the same as that created by Charles Moore. With this VMM, the foundation on which we build Forth software components has been stable since its creation and provides a very good platform on which to build, no matter the underlying processor architecture, register provision or memory space available. The Forth Virtual Machine is compact and easily implemented in incredibly small spaces (~512 bytes to 8kbytes), either fully on the target processor or as an interactive umbilical connection (not an option available with many other environments). With this VM as the basis it becomes almost trivial, on a word by word basis, to conduct compliance testing on each and every Forth word you use and create to support your application.

Most Functional Safety Standards require Certified Compilers and, in other languages, these become very expensive, hard to wield and the outcome is not fully certain despite the claims. However, the key is that it is a results based confirmation with a tough inspection and testing regime applied is just as valid and allowed by IEC61508 for Assembler Code. With Forth being, probably, the best macro assembler in the world, it becomes easy to see the way in which to progress. Using Forth alone does not guarantee that a product will be safe. Its use has to be coupled with a relevent and highly capable Design and Development Process that features full version control and configuration management capabilities.

## The Basis of Certification

Once you have determined your approach to constructing your system to meet the requirements, you need to look at the practice of construction for a robust and certified system. You need a few standards in place to accomplish the task. I have mentioned IEC61508 but that is the over-arching Functional Safety Standard for Electrical, Electronic and Programmable Electronic Systems. It is fairly agnostic about software implementation language. In addition to this you should have a couple of other documents within your development organisation (probably better styled as guides). Some years ago I published the Forth Coding Standard as a public domain document. From this start I know that a few companies have adopted and adapted it to great benefit in making the Forth Source Code more readable. It promotes a more literate style of presenting the source code.

Adopting and following such style guidelines should also help in automating some of the more tedious tasks associated with providing a good quality documentation of the installed system. Tools such as DocGen can help here. Additionally, we have the language standards, such as Forth200X, that we can conform to in order to aid portability of code and/or programmers between Forth using organisations. Standards also aid in wider collaborations.

In Forth Certification the elements of interest are:-

- **The words name** which is a reasonably good identifier
- **The Glossary Text** which becomes the words functional specification of performance.
- **The Stack Comments** which details the input and output parameters
- **The Word Make-up** which is the words that this word uses in order to perform its intended function.

To certify that the word performs its allotted functionality and has no side effects we take the above four items and perform the following:

- **Static Inspection** is a Fagan Style examination to ensure that the apparent intent of the requirements as stated in the glossary text are implemented correctly. Such inspection will also require that the words used in the make-up of this word have been applied correctly. Earlier words should also have been certified in a similar manner.
- **Functional Test** to ensure the word actually compiles and performs as specified. Such testing should explore all pathways of logic within the word.
- **Limits Test** to explore the unwanted side effects a word may have and to ensure that such limitations have been properly documented (in the glossary text). This will ensure that only the right amount of data is pulled from the stack, the stack does not underflow and all logic paths have been exhaustively tested to ensure no adverse behaviours exist.

As most Forth systems are built bottom-up, the above becomes just an extension of the normal test as created philosophy. However, for certification the coder should pass code, he is satisfied with, to others to perform the above three processes. That, though, is just to maintain some independence between the coding stage and the certification steps. Many of the safety standards demand such independence.

## Benefits of a Component Oriented Approach in Forth

Once the component has been coded and certified it may be submitted to a library repository so that it is available for re-use in other projects. If the code is stored as a package along with its certification documentation then the whole library could be considered as tried and tested code that could be used anywhere else so long as its provisions matched the requirements of the new use. Why should Certified Forth code not be as moveable and re-usable as say, an M25 nut fitting onto any other M25 bolt. Such mobility and re-use of precoded components will ease the creation of much larger certifiable systems in the future. The level of documentation for a certified component is higher than for a non-certified component due to fully accounting for each components limitations. This is, though, much like documenting the Maximum Permitted Voltages on an electronic component.

*"We have witnessed hosts of microprocessors and microcomputers marching from cradle to grave, right before our eyes. Languages and operating systems come and go. Even in Forth, which I use to code for a living and write about to entertain, we've seen good work done and disappear, come and go. Have we seen the best yet?"* **[C.H.Ting]**

1. I think we might be about to, and you could and should make that happen.

## References & Notes:-

**[Koopman1]** "The Grand Challenge of Embedded System Dependability" by Koopman P.  given in a panel session at Dependable Systems and Networks 29th June 2011.
<http://users.ece.cmu.edu/~koopman/pubs/koopman11_embedded_dependability_challenges.pdf>

**[Koopman2]** "Better Embedded Software Systems" by Philip Koopman, ISBN 978-0-9844490-2

**[Monforton]** "Good vs. poor documentation: Don't be 'that guy'" by Jeff Monforton 17th December 2013.

**[SEI]** "CMMI Distilled" by Dennis M. Ahern, Aaron Clouse & Richard Turner ISBN 0-321-18613-3

**[Kelly]** **"**Configuration Management: The changing Image**"** by Marion Kelly ISBN 0-07-70977-9

**[C.H.Ting]** "Footsteps in an Empty Valley" by C. H. Ting publishd by Offette Enterprises 1985.