Components for Certification.

Paul E. Bennett IEng MIET
Systems Engineer, HIDECS Consultancy

Abstract

Even the humble hexagonal nut has a data-sheet that describes its functionality, performance factors, interfaces and limits beyond which its continued performance is not guaranteed. Electrical and Electronic Components also have data-sheets that describe their functionality, interfaces, performance and limitations. Why should software be any different? Yet, much of the software in existence has not carried through providing this useful artefact of the rest of the engineering world.

Some consider software as a quite different aspect of creative development and so it has, for many, become a black art for devotees of a specific programming language to get to understand the hieroglyphics that they use. Yet software is being used in a wider range of products, some of which are becoming even more mission, security or safety critical, and sometimes, all three aspects simultaneously.

There have been some attempts at Component Oriented Development[3] with artefacts like .NET, and CORBA. Huge system modelling tools, built mainly for the software industry, have grown up that will churn out code from the model, all without the real feel of whether the model was correct or whether the translation of that model to code was correct. In such circumstances it becomes very difficult to be certain about any assessment of the final products fitness for purpose and absence of hidden faults.

In this paper we will take a look at what is required for Component Oriented Development that can be proven to be fully trustworthy to perform as its data-sheet implies.

1 What is a component?[4]

Across all engineering disciplines, the author considers that the following features should be common to all components. In fact Components

- have a unique reference identifier
- have Surfaces by which other components are interfaced.
- have been specified for operation within given environmental constraints
- have data-sheets that describe all functionality, features, performance and limitations of guaranteed performance.
- can be used and re-used many times over.
- can be inspected, tested and certified individually without impact on other components.
- conform to standards relevant to its functionality and performance.
- can, once certified, be used without being re-certified for every new situation, provided the new situation does not exceed the expectations of its published data-sheet.

However, using a certified component will not imply that the whole system is certified just by using it. To certify a whole system, the whole system needs to be constructed from known certified components throughout, have an audit trail that has logged all component certification, and itself be tested against its own statement of requirements.

For software components, there is a need for a development environment that allows the easy inspection and testing for individual components, preferably without having to write special test stubs to implement the testing. Where test stubs have to be created to perform

the test these should be logged with the component for subsequent confirmation testing and should receive as much attention to their correctness as the component itself.

2 Component Specification

Specifications of components grow out of the specification of the system to which they will ultimately belong. Such specification will mention aspects like the operational environment, lifetime expectations, MTBF (Mean Time Between Failures), Maximum and Minimum expectations of operation, Nominal Operating Regions and perhaps some notes on intended methods of use. Specifications, whether for the entire system or just a single component, should always adhere to the precept that they are Clear, Complete, Concise, Coherent, Correct and Confirm-able. Any lesser adherence to the principal 6 Cs[5] of specification will detract from the ability to fully assess the quality and robustness of the eventual product.

3 Component Management

Having created a component, all the artefacts, such as designs, data-sheets, inspection, test reports and other ancillary information relevant to the components use (like application notes) should be stored in a secure archive for which there is strong version control and strict change management procedures in place. This ensures the longevity of information about the component and its inspection and testing.

Regular auditing of the archive ensures that versioning and change management processes are being carried out properly and that the security of the information remains unsullied. The version control and change management becomes a very important aspect to development processes where the expected outcome of a development is a safe, secure, mission critical product.

4 Component Inspection, Testing and Certification

The Requirements of High Integrity Systems, especially in the Safety Critical[1] world, are:-

- Arg1 - the system has been specified to be safe - for a given set of Safety Criteria, in the stated operational environment
- Arg2 - the resulting system design satisfies the agreed specification
- Arg3 - the implementation satisfies the system design

In examination in accordance with these three arguments, those who inspect the component (and system) will need to see robust evidence that the material presented is valid. Such demonstration is given by provision of:-

Direct evidence - which provides actual measures of the attribute of the product (i.e. any artefact that represents the system), and is the most direct and tangible way of showing that a particular assurance objective has been achieved.

Backing evidence –which relates to the quality of the process by which those measures of the product attributes were obtained, and provides information about the quality of the direct evidence, particularly the amount of confidence that can be placed in it.

The references to inspection and testing, above, have specific connotations in the light of components. For the mechanical world, there will be certificates on the material being used to assure that it is of the appropriate quality for the intended purpose. Physical viewing of the component to confirm its identity as the right component for the task, and measurements of

the final component to ensure that it conforms to its design data (as in the case of the nut, checking all the components dimensions to ensure a match to the drawings). There may even be a destructive stress test conducted on a small sample of the component to ensure the design criteria has been met.

For software, whilst we will still need an inspection and testing method to ensure that the design criteria is met, the methods are slightly different. Below, we will cover the three aspects of inspection and testing, namely the Fagan Style Inspection, Functional Testing and Limits Testing.

4.1 Inspection of Software

The author recommends the Fagan Style Inspection[2] as the best technique to perform a rigorously intense examination of the software itself. Getting to the point of inspecting a component for certification will have already initiated a series of inspections and reviews to ensure that the specifications on which the specification of this component relies are sound in principle and capable of compliance. Aspects that need to be observed during this inspection are:-

- Each component shall have a full statement of specification in which the functionality, performance characteristics, methods and limitations of the software component are fully described (references to specific clauses in standards or other document relied upon for the component are permitted but have to be made available to the inspection team).
- Any components on which this component relies already has certification in place as evidenced by the availability of that components certificate of conformity.
- All logical pathways through the code are checked individually to ensure that there are ways in which all pathways can be executed, and that the logic used is sound. Preferences are for simple decision structures or non decisions at all.
- The logical pathways in the code implement precisely the logic demand by its specification. Disparities should be recorded in the inspection and test report and regarded as a failure.
- The component has exactly one entry and one exit point.
- The Cyclomatic complexity is as low as is reasonably practical to the intended task described in the specification.

As you will detect, a lot of reliance is placed on having the specification and code closely allied during the inspection process. Fagan Inspections are, essentially, a style of static analysis but conducted with close attention to the details of implemented intent.

4.2 Functional Testing[6]

Functional testing, for certification, has to operate the component in its normal mode function but ensure that all logical pathways are fully exercised. The requirement is 100% logical pathway coverage. Running a functional test with a copy of the source code to hand and a marker to indicate when the pathway is taken and under what conditions. The function performed should precisely match the description in the component specification. Any deviation from the functional specification is seen as a failure of the functional test and should be noted in the test report.

4.3 Limits Testing[6]

Much of software may operate without encroaching any limitations whatsoever.

However theoretical the limitless possibilities might be, all implementations of a software component will exhibit limits with respect to the cell-width of the machine on which it will operate. So long as such limitations are understood by the user of the component there is usually no real concern.

However, some software components implementing specific algorithms, will exhibit a limitation of their accuracy or performance outside certain bounds. Hence, the specification should make it clear where such limits theoretically lie in order that testing against such limits can be undertaken to ensure the component continues remains to remain stable despite exceeding such limits (ie: takes the appropriate actions when limits are exceeded). An example of such a limitation is the divide by zero error in routines that use division. For such errors, an appropriate means of managing the error needs to be put in place and tested to ensure that in all cases where the limitation is achieved, the proper course of action is always taken.

Implementing such testing often requires quite wild imaginations to accomplish but the intention is to actively try and destroy the software component, much like you would destroy the test sample of a mechanical component.

5 Summary

This paper has been but a brief run-through of the Component Oriented approach to software development. We have briefly mentioned the need for all components to have a data-sheet in which its functionality, interfaces, performance and limitations are fully described. Additionally, we have covered a brief overview of the necessary inspection and testing regimes by which component certification can be accomplished. Treating the development of software components similarly to the development of any hardware component, with a specification, inspection, and testing regime that is fully explorative of the component properties, will improve overall quality of the delivered system. Finally, that attention to detail is beneficial to the outcome and re-usability of the components developed by this means.

That the above implies an increase in documentation should not be seen as any reason to reject such an approach, as this increase in documentation is substantiated by the ease with which certification of components can be achieved.

This usually leads to an eventual saving of development costs for those developing the higher integrity systems which will ensure our continued safety and security.

6 References

[1] **Functional Safety by Design – Magic or Logic?** Derek Fowler; Safety-critical Systems Symposium, Bristol UK, February 2015.

[2] **A History of Software Inspections** Michael Fagan, sd&m Conference 2001, Software Pioneers Eds.: M. Broy, E. Denert, Springer 2002 <http://www.mfagan.com/pdfs/software_pioneers.pdf>

[3] **Component Software** <http://www.webopedia.com/TERM/C/component_software.html>

[4] **Component** <https://en.wikipedia.org/wiki/Component>

[5] **High Integrity Systems CODE** Paul E. Bennett IEng MIET, EuroForth 2014.

[6] **Software Testing – Goals, Principles, and Limitations** S.M.K Quadro & Sheik Umar Farooq International Journal of Computer Applications <http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.206.4616>

The author is willing to advise on and oversee any software component oriented development process leading towards full system certification against any standard. He is also a member of the IET, Safety Critical Systems Club and has been using Forth, in a component oriented manner, for high integrity systems since the mid 1980's.

Paul E. Bennett IEng MIET <Paul_E.Bennett@topmail.co.uk>

Tel: +447822639972