DETERMINING THE INTEGRITY OF COMMERCIAL SOFTWARE IN ADVANCED AVIONIC SYSTEMS

Mike Lane, Sensor and Avionic Systems Department, Defence Evaluation and Research Agency (DERA), Farnborough, Hampshire, UK

Abstract

Rotorcraft platforms are becoming increasingly dependent upon electronic systems controlled by software to meet functionality requirements. This is further driven by requirements to reduce platform weight and complexity by replacing hydraulic and mechanical systems, where possible, with electronics. The result is that avionics can now account for over 40% of the through-life costs of rotorcraft platforms. The requirement to support increasingly demanding application functions is such that this figure, in both absolute terms and as a percentage of the total system cost, is set to increase for new platforms and upgrades to existing systems. If these costs are to be contained avionic systems will become increasingly dependent upon COTS (Commercial Off The Shelf) components for both hardware and software. Although these products may appear to meet real-time performance requirements individually their integration and certification in aircraft systems requires rigorous proof of correctness. The problem of certification of systems with COTS components is exacerbated by the rate at which new versions of the products are released, which may not be compatible with other system components. The UK Ministry of Defence (MOD) has therefore funded research into the integration of COTS components into advanced avionic systems.

While the use of commercially available hardware components has, to some extent, been accepted as the only way forward, the use of COTS software components has been highly contentious. Although the potential benefits can still apply to software, new challenges are introduced that must be overcome. Although software does not fail in the same way as hardware, the effect on system behaviour can be just as serious, and the resulting costs even higher. These problems are exacerbated by the inherently integrated nature of advanced avionics. A key area of concern for integrating commercially available software products into high integrity systems is proving that the software will be both safe and reliable. These areas have been tackled differently and the results so far are summarised in this paper.

The very idea of trusting COTS software in a complex real-time system that may affect, or even be responsible for, safety critical or mission critical functions has been the subject of much debate. The concerns have mainly been centred on dependability and safety certification. Although an avionic system will include a large number of software components of different types, a particular problem was identified with operating system certification. It is felt that this will have the greatest impact on affordability and flexibility, and presents the greatest challenge for certification and re-certification of the platform. This is not only a very complex component, but also one that has commercial support from a range of vendors. The operating system is also a component likely to require upgrading as hardware resources are themselves upgraded to take advantage of commercial developments.

Until a few years ago it seemed unlikely that it would be possible to meet UK military certification requirements for a platform using a COTS component as complicated as an operating system for mission or safety critical functions. Even if it was possible, it seemed that the costs involved would be prohibitive and the traditional approach of developing bespoke, ie non-COTS, solutions would be more cost effective. This has been a major stumbling block in their acceptance and has often led to bespoke solutions. Since that time, however, operating system vendors have taken this problem seriously. Some have produced documentation and proofs to support certification, and several have even achieved some level of certification certification.

Aircraft certification, of course, applies to a complete system rather than individual components. For an avionics system therefore, the safety case must form part of the overall aircraft safety case. Given, however, that the move to advanced avionic concepts will support regular upgrades, the generation of a full safety case for each upgrade would not be practical. A modular approach is therefore proposed for the safety case itself, this is discussed in the paper.

In addition to showing that COTS software components can meet safety requirements, they must also be shown to be reliable and maintainable throughout the life of a helicopter platform. Software failure prediction techniques have been used across many application domains, and software reliability modelling is now a highly developed area in software measurement. The results of research to determine the applicability of these techniques for avionics software are discussed, again with emphasis on the real-time operating system (RTOS) software.

Introduction

Whilst the functional requirements and cost of helicopter avionic systems have increased rapidly relative to overall platform costs, the defence industries proportion of the electronics market has fallen to under 1%. In order to exploit the huge investments and developments aimed at the commercial market, military systems will become increasingly dependent upon COTS (Commercial Off The Shelf) components for both hardware and software. Although these products may appear to meet real-time performance requirements individually their integration and certification in aircraft systems requires rigorous proof of correctness. The problem of certification of systems with COTS components is exacerbated by the rate at which new versions of the products are released, which may not be compatible with other system components.

The UK Ministry of Defence (MOD) has therefore funded Sensor and Avionic Systems Department (SASD) of DERA to conduct research into the integration of COTS components into advanced avionic systems. Although helicopter systems are a key exploitation route for this programme, it is relevant to all airborne platforms that rely on processing systems to complete missions effectively. The study has been supported by Praxis Critical Systems who have provided expertise in the assessment of systems for high integrity applications.

A key area of concern for integrating commercially available software products into high integrity systems is proving that the software will be both safe and reliable. These areas have been tackled differently and the results so far are summarised in this paper. Although many of the results may be applicable to a wide range of software components, the focus for the research has been the operating system. It is felt that this will have the greatest impact on affordability and flexibility, and presents the greatest challenge for certification and re-certification of the platform.

Safety and Reliability Definitions

Although safety and reliability requirements may be related, they clearly cannot be treated as identical. While it may seem likely that a highly reliable system will be safe and therefore easy to certify, and visa versa, there is no guarantee of this in practice. In fact, there are plenty of examples of systems that are very reliable but considered to be unsafe, and also examples of systems that exhibit very low reliability but that are safe. Both of these characteristics have therefore been addressed.

For software components the ANSI definition of reliability is '*The probability of failure free software operation for a specified time in a specified environment*'. For the purpose of this study a failure is assumed to be a deviation from the *intended* specification, not simply a deviation from a specification which itself may be incorrect.

Safety[1] is defined in terms of the features that have been implemented in order to reduce to an acceptable level, or even eliminate, the risk of a hazard occurring. A safety requirement can therefore be described using a desired behaviour and a set of conditions under which this behaviour is safe. *Safety integrity* is defined as the probability of a safety-related system satisfactorily performing the required safety functions under all stated conditions within a stated period of time.

Safety integrity is composed of two components, *systematic safety integrity* and *random safety integrity*, where:

- Systematic safety integrity is based on the claim that the system is free from errors, that under a particular set of conditions will cause a failure.
- Random safety integrity is based on the claim that for the stated period of time the system will not suffer from failure caused by random failure mechanisms in the hardware.

Table 1. Safety Certification Standards

Standard and Definition of Safety Integrity	**Definition of Safety Integrity Levels**
DEF STAN 00-55 (UK military standard for avionics) The likelihood of a system achieving its required safety features under all the stated conditions within a stated measure of use.	SIL is an indicator, assigned in accordance with Def Stan 00-56, of the required level of safety integrity. There are four safety integrity levels defined, S1, S2, S3, and S4, where S1 is the least critical and S4 is the most critical (safety critical).
RTCA D0-178B (US standard for avionics) Does not define the term.	Mentions **Software Levels**. A software level is based upon the contribution of software to potential failure conditions as determined by the system safety assessment process. The software level implies that the level of effort required to show compliance with certification requirements varies with the failure condition category. The defined software levels are **Level A**, **Level B**, **Level C**, **Level D**, and **Level E**, where Level E is the least critical and Level A is most critical.
IEC-61508 (International standard for industrial control etc. Some tailoring is expected if it is to be applied to avionics) **Safety integrity:** Probability of a safety-related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time. **Software safety integrity:** measure that signifies the likelihood of software in a programmable electronic system achieving its safety functions under all stated conditions within a stated period of time. **Systematic safety integrity:** part of the safety integrity of safety-related systems relating to systematic failures in a dangerous mode of failure.	**Safety integrity level:** discrete level (one out of a possible four) for specifying the safety integrity requirements of the safety functions to be allocated to the E/E/PE safety-related systems. The defined levels are **SIL1, SIL2, SIL3,** and **SIL4**, where SIL4 has the highest level of safety integrity and SIL1 has the lowest. **Software safety integrity level:** discrete level (one out of a possible four) for specifying the safety integrity of software in a safety-related system.

The definitions of safety integrity and safety integrity levels vary slightly across the various standards. Table 1 gives the definitions provided by DEF STAN 00-55[2], RTCA DO-178B[3], and IEC 61508[4].

For the purposes of this document the terminology of Def Stan 00-55, which defines the requirements for safety related software in defence equipment, is used.

Software Integrity Assessment

It appears likely that some COTS software products can meet cost and real-time performance requirements for avionic systems. The issue of demonstrating reliability and safety, however, has not proved to be as straightforward. These two areas have therefore been considered separately, although the basic question remains the same: 'Is it both possible and worthwhile to use complex COTS software components in avionic systems for future aircraft and upgrade programmes?'

Software Safety

Until a few years ago it seemed unlikely that it would be possible to meet UK military certification requirements for a rotorcraft, or any other airborne platform, using a COTS component as complicated as an operating system controlling mission or safety critical functions. Even if it was possible, it seemed that the costs involved to prove this would be prohibitive and the traditional approach of developing bespoke, ie non-COTS, solutions would be more cost effective. Since that time, however, operating system vendors have taken this problem seriously. Some have produced documentation and proofs to support certification, and several have even achieved component certification. The feasibility of achieving platform certification for a system with complex COTS components is the subject of a UK MOD research programme.

Whilst pseudo-operational testing and analysis can be applied to systems with reliability requirements in the order of 1×10^4 hours, safety related systems require a reliability in the order 1×10^8 hours. Although the random element of this can be shown using numerical analysis, the systematic element cannot be measured reliably for such small probabilities. Most current safety standards address this problem by assigning safety integrity levels (SILs) to safety related functions (see Table 1). The higher the integrity level, the greater the degree of rigour required in developing the software. The SIL, however, is not the only requirement for certification. It is this point that has caused some confusion with respect to the certification of the *implemented* software component. Some software developers have claimed that their software is safe because it has been developed in accordance with processes commensurate with the assigned SIL. While this may go a long way to achieving certification, is not the entire requirement. Successful certification is also dependent upon the integrity of the final executable, which may contain components not covered by the defined process. Reference will also need to be made to the specific hardware on which the software executes and integration with other software components.

The Software Safety Case (SSC) provides a means of presenting all arguments required for safety certification. The issues surrounding the development of the SSC for avionic systems containing COTS software products has been the subject of this research. Although the avionic system will include a large number of software components of different types, a particular problem was identified with operating system certification. This is, potentially, a very complex component and one that has commercial support from a range of vendors. The operating system is also a component likely to require upgrading as hardware resources are themselves upgraded to take advantage of commercial developments.

In order to assess the safety of commercial operating systems in avionic systems the following questions must be addressed:

- What kinds of operating systems are available commercially?
- How can we determine the level of integrity required for a Real-Time Operating System (RTOS)?
- Can a commercial off the shelf RTOS demonstrate this level of integrity?
- How much certification support is available from the RTOS vendor throughout the lifetime of the product?

There are a number of commercially available RTOSs that have been specifically developed for high integrity application. Examples are OSE from ENEA OSE Systems, VRTX from Mentor Graphics Corporation, QNX from QNX Software Systems Limited, LynxOS from Lynx Real-Time Systems, and RTEMS, which was developed under contract to the US Army Missile Command. These have varying levels of certification track record. It is not clear, however, whether any of these COTS RTOS will be able to demonstrate compliance with the highest level of integrity.

The operating system providing the focus for this study is OSE. This contains 70 system calls of which 30, the safe kernel, have been certified to S3 in accordance with IEC 61508.

In order to achieve certification proof must be provided in the safety case that the system meets its integrity requirements. This proof would be based on the application of rigorous development techniques and evidence obtained from verification and validation activities, which should be subject to independent assessment and audit. As already stated, the certification applies to a complete system rather than individual components. For an avionics system therefore, the safety case must form part of the overall aircraft safety case. Given, however, that the move to advanced avionic concepts will support regular upgrades, the generation of a full safety case for each upgrade would not be practical. A modular approach is therefore proposed for the safety case itself. The arguments for components of the system will be presented as self-contained elements allowing the reuse of existing elements for re-certification in the case of a component upgrade. The component safety arguments should contain:

- component safety requirements for the specific avionic application or applications.
- the argument for and evidence that the safety requirements have been met.
- restrictions and guidelines that apply to the safe integration of the component into the system.

Specifically, for an RTOS, the safety case should include:

- **Scope.** To contain a description of the system in which the software operates, record any assumptions for the arguments made, and discuss updates and change control.
- **Software description.** To discuss the operational behavior of the RTOS.
- **Software safety requirements.** To discuss the safety requirements on the RTOS as part of the avionics application, including how they are derived and defined.
- **Software architecture.** To describe the RTOS architecture, overview of protocols used, essential operations, etc.
- **Software development process.** To provide a description of the RTOS development process and the tools that have been used during this process.
- **Safety arguments.** Arguments should be presented to demonstrate that the risk of an error in RTOS software leading to a hazard is acceptable.
- **Discussion of issues.** Any outstanding issues that affect the integrity of the RTOS should be identified.
- **Conclusions.** This provides an engineering judgement as to whether the risk of an error in the COTS RTOS leading to a hazard is within acceptable limits.

Initial research indicated that some operating system vendors would be able, and willing, to provide evidence in support of such a safety case. However, given that none had yet achieved the highest level of product certification it was clear that further assurance was required. It was decided that this should be addressed by sending a team of industry and DERA experts to the software development headquarters of a real-time operating system vendor. An assessment plan was developed and ENEA OSE Systems agreed to facilitate the visit, and make sure that senior software engineers were available to answer questions and provide detailed evidence of software development and verification processes.

The results of this evaluation are now being used to develop a 'draft' safety case for a high integrity application executing under the control of a commercial RTOS. Given the scope of the study, which is aimed at evaluating the practicality of such an approach to system design rather than full system development, a single application has been selected. This is a Stores Management System (SMS) which is considered to be complex enough for proof of concept but within the scope of a research programme.

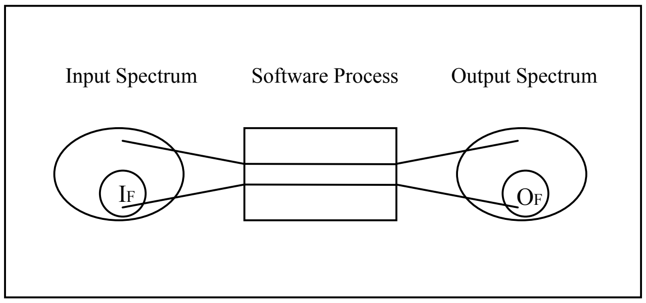
Whilst considerable progress is being made by operating system vendors to meet the requirements of mission critical and safety critical systems, and MOD research is developing methods and techniques to support the introduction of COTS software, there are still questions to be answered. These centre on the questions of not only the technical viability of the concept for initial system development, but how systems made up of COTS components will be maintained and upgraded. It is therefore intended that these issues will be addressed as part of the ongoing programme

Software Reliability

Most current system reliability analysis is centred on the reliability of the hardware, but ignores software, effectively assigning it a reliability of 1. This is further complicated by the fact that for certain certification activities, software is attributed a reliability of 0, ie no credit is given for the integrity of software products. This obviously provides no help in answering the real question, ie 'when can software be accepted for system implementation, not just for safety critical but in all applications?' The system designer wants primarily to be able to measure the reliability of the software subsystem at a given time. This can then be compared with the reliability target to decide whether the software is fit for release. However, it is also a requirement to predict the level of reliability that will be achieved following a given amount of testing effort. It became clear that techniques with a more scientific basis were needed to predict software behavior more accurately and to mitigate the effects of software faults.

Although software failure mechanisms differ from those experienced with hardware they can both be described by probability distributions. However, while some electronics failures can be modelled using constant failure rates, i.e. an exponential reliability distribution,

Figure 1. Software execution model



the failure rates of software cannot be assumed to be random. Although the individual times to failure may be random, there is usually a trend. For example, if errors are detected and either corrected, or the software used in such a way as to minimise the probability of these errors occurring, the reliability will tend to improve. If, however, maintenance actions are carried out without carefully considering the side effects, although one fault may be fixed others may be introduced. The reliability of the software may therefore deteriorate. Software failures must therefore be modelled as a non-linear, non-stationary process.

The basic software execution model used is shown in Figure 1.

The input spectrum refers to all possible combinations of inputs to a software process. A subset of these, I_F, will produce unacceptable outputs, i.e. O_F. The objective of software reliability analysis is to determine the likelihood of such an event, or combination of events, occurring.

The two main types of model are prediction models and growth models. Prediction models use parameters associated with the software product and the development environment, for example language, complexity, level of reuse, size, and the level of interconnection of the compilation units, to make judgements regarding reliability. The technique requires a 'proof program', or baseline, for comparison. The criterion for this baseline is similarity, which often cannot be achieved. Given the diversity of software in avionics it was felt that such a baseline would not be available, and research focussed on growth models.

Growth models, sometimes called parametric models, use failure data from the application being modelled to determine parameters affecting reliability. This is used to estimate the probability that a system will not fail during a given time period.

The key requirements and assumptions for applying growth modelling techniques to software are that:

- The failures are randomly distributed across the set of possible inputs. This can generally be assumed if we do not know the inputs (see Figure 1) that will cause failures or the order in which they will be selected from the input space.
- Actual data from the software being executed in a representative environment is available.
- Extensive software changes are not routinely made while data is collected.
- All software changes to correct faults are perfect, ie a given fault will not occur twice. In the case of COTS software components it may not be possible to make such changes immediately. In this case either the knowledge of the fault may be used to control the selection from the input space, or later manifestations of the fault could be discounted for modelling purposes. This is not identical, however, as faults that would have been unlocked by fixing other faults will not now show themselves.

Numerous models and tools are available for growth modelling, each with their own mathematics. These can be categorised depending on, for example, assumptions of whether a finite or infinite number of faults will occur in an infinite time, the distribution of cumulative failures, and the failure intensity function in terms of time (eg Weibul, exponential, or gamma).

Examples of these models[5,6] are Certification, Geometric, Goel-Okumoto, Hyperexponential, Jelinski-Moranda, Littlewood-Verrall, Musa-Okumoto, Schneidewind, and Shooman. It is not the aim of this paper to go into details of specific models, but to indicate how they could be applied in the real world. It became clear that no single model was universally applicable, and that even if a model appeared to work very well for a particular application there was no guarantee that this trend would continue. It would be very likely that a model that performed very well would become out-performed by a model that earlier in the assessment had performed badly. The three forms of inaccuracy are noise, consistent bias, and non-stationary bias.

Until recently this made many models unusable and the confidence in the whole approach of growth modelling was affected. However, solutions have been developed to alleviate these problems. These are now summarised:

Prequential Likelihood (PL)[6] and the Prequential Likelihood Ratio (PLR) enable the comparison of the performance of several models for a given data set. The PL is a measure that denotes a model's accumulated accuracy. This is calculated for each model that has been applied to the software being assessed, and the ratio determined for any two models. This will show which of them is performing best. Repeated calculations of ratios for other models and the 'best' found so far will give the most suitable model for the data available.

While PLRs allow comparisons to be made between the performance of models they do not tell us anything about the true accuracy of a model. U-plots provide a method to determine whether the predictions, on average, are close to the true distributions. While the u-plot will provide a measurement of consistent bias, a further technique called the y-plot can be used to detect non-stationarity.

It is hoped that by eliminating unsuitable models a model will be left that provides fairly accurate predictions. However, in many cases it turns out that none of the models can provide the required accuracy without further intervention. A comparatively recent development is recalibration. This technique aims to detect bias in a model's predictions and feed an appropriate compensation factor back into the model so as to remove this bias. We are therefore trying to find a function G_i that would transform the existing prediction, $\hat{F}_i(t)$, onto reality, $F_i(t)$. Therefore:

$$F_i(t) = G_i[\hat{F}_i(t)]$$

This can provide a significant improvement in the predictions being made, even when the predictions have some degree of non-stationarity. Tool support is currently being developed for recalibration.

Applicability to COTS

The problem with modelling the reliability of COTS software[7] is the typical lack of knowledge regarding the structure of the component, or underlying factors that determine its reliability. As far as reliability analysis goes COTS does not form a software classification in its own right. Indeed there are a variety of techniques that apply for commercial software, influenced by the way that the software is used and its basic function.

In the case of software with an accurate and complete in-service history, then it would be possible to provide a prediction for the Mean Time Between failures (MTBF). It is also essential to understand the operational profile of the history in order to relate this accuracy to any new application.

For commercial software components without a clear history, pseudo-operational testing would be required to estimate reliability. This would be in addition to any validation or conformance testing. If the test regime is strictly in accordance with the intended operational profile of the system then there is no reason why parametric modelling should not be used to provide good results. These techniques could then provide confidence in MTBF up to the order of 1 x 10^4 hours. (Note – this figure relates to what is practically possible rather than theoretically possible). However, if acceptance testing is more of a continuation of system testing, ie to find faults even if they would not have caused an operational failure, then other modelling techniques would be more suitable.

Where applications require confidence greater than an order of 1 x 10^4 hours, eg safety related functions, software reliability modelling techniques alone would not be enough to achieve certification. It is, however, possible to build commercial components into a higher integrity application using techniques such as safe partitioning and fault tolerance. This has already been achieved successfully in federated avionic systems and has much scope for further research.

Conclusions and Future Studies

In addition to showing that COTS software components can meet functional and performance requirements, they must also be shown to be reliable and safe. This has been a major stumbling block in their acceptance and has often led to bespoke solutions. These areas have therefore provided the focus for research into the application of COTS in avionic systems. It is important to appreciate that although COTS products present their own challenges, bespoke software is not immune from problems of safety and reliability acceptance. Also, bespoke products will not benefit from the wider usage and very large user base of their commercial counterparts.

The increasing dependence of rotorcraft systems on complex software is such that software safety certification is a growing problem. Although the study has been primarily aimed at commercial RTOSs, many of the techniques, and the expertise, developed are applicable to other software components. The approaches should therefore be applied to address other areas with the potential to utilise COTS technologies, and where proof of integrity has been identified as a potential problem area, such as automatic system reconfiguration.

Experiences gained from the integration of COTS software products in complex, high integrity systems should be closely monitored with reference to cost, reliability, and performance. These lessons will be invaluable for determining the most suitable approach for using commercial solutions for avionic upgrades and new systems.

With respect to software reliability, the development of models and tools to predict reliability of software has been the subject of extensive research in academia, government, and industry and should not be ignored. Although a general solution has not, and will probably never be found for COTS, the real issue that must be considered is 'what is the alternative?' A more scientific approach to determine if and when software can be accepted rather than assuming that the most recent correction will be the last must be advantageous. The techniques studied can provide this scientific basis and have been used successfully for some major projects.

The potential benefits of the research will only be achieved if clear exploitation routes are determined and followed up. Several helicopter programmes have already been identified and future research activities will take into account specific constraints and requirements of these systems.

Acknowledgements

The author would like to acknowledge Praxis Critical Systems for their support throughout this project.

References

1. Raili Efi, Alan Simpson, April 2001, DERA *COTS Safety, Reliability and Certification Study – Final Report*.
2. Def Stan 00-55, *Requirements for Safety Related Software in Defence Equipment*.
3. RTCA DO-178B, *Software considerations in Airborne Syatems and Equipment Certification.*
4. IEC 61508, *Functional Safety of electrical/electronic/programmable electronic safety-related systems.*
5. Simpson, Alan, Iain Lees, February 1997, Contract Report - *Software Reliability Study by Praxis Critical Systems*.
6. Lyu, Michael R, 1996, *Handbook of Software Reliability Engineering*, New York, McGraw-Hill, Chapters 2-4.
7. Simpson, Alan, Iain Lees, February 1997, Contract Report - Software Reliability Study by Praxis Critical Systems.
8. Randall, B, C Laprie, H Kopetz, Littlewood B, 1995, *Predictably Dependable Computing Systems*, New York, Springer,
9. Wilcock, Geoff, Terry Totten, Alan Gleave, Roger Wilson, April 1999, '*The application of COTS technology in future modular avionic systems'*, Journal of Defence Science.